Une nouvelle stratégie de glissement de données pour les caches élastiques dans les architectures manycoeurs.

Safae Dahmani

DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

Département
Architecture, Conception et Logiciels Embarqués

Saclay, le 14 septembre 2012

REF : LIST/DACLE/DRT/LIST/DACLE/12-0648/S-SD

Une nouvelle stratégie de glissement de données pour les caches élastiques dans les architectures manycoeurs

par
Safae DAHMANI

DRT/LIST/DACLE/LaSTRE (CEA)

list
Laboratoire d'Intégration des Systèmes et des Technologies

leti
Laboratoire d'Electronique et de Technologie de l'Information

Commissariat à l'Energie Atomique et aux Energies Alternatives
Institut Carnot CEA LIST
Centre de Saclay | Nano-Innov Bât 862 | PC 172
91191 Gif sur Yvette Cedex
Tel. : +33 (0)1.69.08.49.67 | Fax : +33(0)1.69.08.83.95
thierry.collette@cea.fr

Direction de la Recherche Technologique
Département Architecture Conception et Logiciels Embarqués

Première partie

Remerciements

Ce document est un rapport de stage de recherche effectué dans le cadre du Master Recherche en Systèmes Embarqués et Techniques d'Information de l'ENS Cachan (Ecole Normale Supérieure de Cachan). Le stage s'est déroulé au CEA Nano Innov, au sein du Laboratoire des fondements des systèmes temps-réels embarqués(LaSTRE).

Le sujet de stage s'intitule : Une nouvelle Stratégie de glissement de données pour les caches élastiques dans les architectures manycoeurs. Il s'est déroulé sous la responsabilité de LOÏC CUDENNEC, ingénieur chercheur au CEA.

Je remercie mon encadrant LOÏC CUDENNEC, l'ensemble des membres du laboratoire pour l'accueil qu'ils m'ont réservé. Je remercie également JEAN THOMAS ACQUAVIVA, ingénieur chercheur au CEA, pour son aide portant sur le Cache Validator dans la partie expérimentale de mon stage. Je tiens à remercier particulièrement mes collègues du bureau pour leur soutien et leur convivialité.

Table des matières

Deuxième partie

Contexte général

1 Evolution des systèmes de calcul numérique

1.1 Les premiers calculateurs numériques

A la fin des années 30, l'histoire des ordinateurs a vu apparaître plusieurs prototypes fonctionnant en binaire et basés sur la logique booléenne. Ensuite, il y a eu la création de la machine entièrement électronique ENIAC (Electronical Numerical Integrator And Calculator), premier calculateur moderne. En 1945, Von Neumann propose de stocker le programme à exécuter dans la mémoire de l'ordinateur. Ceci constitue une avancée technologique, en offrant l'automatisation de la programmation difficile de la machine ENIAC, effectuée manuellement par connexion de 6000 commutateurs.

Depuis, l'apparition des circuits intégrés vers la fin des années 60, l'industrie des ordinateurs voit une montée en puissance de calcul et en miniaturisation. Ce qui donne naissance au modèle Intel 4004 en 1971, qui serait ainsi la première unité de calcul intégrée entièrement sur une seule puce.

Au début des années 80, le monde voit l'apparition de la technologie CMOS, ainsi une nouvelle génération d'ordinateurs voit le jour. La technologie CMOS va tout de suite s'imposer comme technologie dominante de l'époque. Ensuite, viendra le lancement des ordinateurs dits Personal Computer par IBM, ce qui va avoir un grand impact sur l'essor de cette industrie. Cependant, les performances des systèmes continuent à évoluer grâce à l'évolution des technologies CMOS pour les microprocesseurs et les mémoires RAM d'un côté et des technologies flash SSD pour le stockage de masse d'un autre côté.

1.2 La naissance des superordinateurs

Les besoins en puissance de calcul des applications scientifiques ont contribué depuis très longtemps à créer une classe particulière de machines, appelée superordinateurs, caractérisée à la fois par une performance très supérieure à celle des gros ordinateurs et par un coût d'un ordre de grandeur plusélevé. Les superordinateurs étaient initialement des machines vectorielles à un seul processeur super puissant. La naissance du parallélisme d'instructions est une approche qui permet la réduction du temps d'exécution de chaque programme individuel. Or, s'il est relativement simple d'acquérir et traiter quatre instructions par cycle, on ne peut de manière triviale passer à huit, seize, etc. De ce fait, les systèmes massivement parallèles avec implémentation de plusieurs processeurs sur une même machine apparaissent. Une technique qui facilite l'augmentation du débit d'exécution sur un ensemble de programmes et non seulement sur un seul.

Le processeur ILLIAC pleinement opérationnel en 1976, est l'un des premiers ordinateurs destinés au calcul massivement parallèle. Il utilisait jusquà 256 processeurs. D'autres familles de superordinateurs succèdent la série ILLIAC, notamment la série de machines Cray fabriquées dans les années 80, la famille ASCI Red d'Intel à la fin des années 90 ou encore la génération Blue Gene d'IBM sorties un peu plus tard. A l'heure actuelle, en fonction de la taille des problèmes, machines vectorielles et machines parallèles extensibles se partagent le marché.

1.3 Évolution des systèmes vers les manycoeurs

La loi de Moore, fruit d'une extrapolation empirique, prévoit la montée exponentielle en puissance des ordinateurs en se basant sur l'observation de la croissance de la technologie à semi-conducteurs et son faible coût. Il existe plusieurs variantes de l'interprétation de la loi de Moore. Notamment, la version sur la fréquence d'horloge qui semblait suivre les prévisions. Cependant, en raison de difficulté de dissipation thermique, la fréquence d'horloge de processeurs stagne depuis 2004. La loi sur la montée en fréquence n'est donc plus vérifiée.

Une deuxième variante de la loi de Moore, concerne la densité des transistors sur puce. En effet, La fréquence restant pour sa part inchangée, le nombre de transistors sur puce double tous les 18 mois. Cette version de la loi de Moore est aujourd'hui encore vérifiée, elle donne naissance à la génération supercalculateurs à plusieurs coeurs dits multicoeurs.

Il s'agit de processeurs réunissant plusieurs instances de calcul dans un même composant. La logique demeure la même que pour les machines à multiprocesseurs : augmenter la capacité de calcul en utilisant des unités logiques (CPU) supplémentaires. Dans les systèmes multicoeurs, il ne s'agit plus d'une multiplication du nombre d'unités de calcul mais d'une division de celles existantes.

L'approche multicoeurs correspond à une demande grand public pour amélioration des performances (exp : Applications multimédia, Jeux) en dépit de la limitation de l'augmentation en fréquence.

Actuellement le marché propose des systèmes à plusieurs dizaines de coeurs. Quelques exemples commercialisés sont traités plus tard dans le chapitre 3. Toujours contraint par la dissipation thermique, le facteur coût et principalement le besoin en performances ; la tendance actuelle est de passer à une échelle plus grande en nombre de coeurs par noeud de calcul. Conséquemment, les prochaines puces à sortir au marché comporteront quelques centaines de coeurs de calcul. Les fondeurs commencent donc à mettre en place des plateformes de type manycoeurs (exemple : le processeur MPPA-256 de Kalray).

L'idée derrière les architectures de type manycoeurs est d'augmenter le nombre de coeurs tout en gardant des fréquences réduites. Une solution qui permet de maintenir le contrôle du coût de consommation électrique et de la dissipation thermique. Cela permet également de diminuer l'écart entre le débit de traitement des instructions et la latence engendrée par les accès à la mémoire et les communications réseaux, grâce à l'utilisation de plusieurs niveaux de caches par coeurs.

Aujourd'hui, le développement des systèmes hautes performances pose des problèmes de consommation énergétique. Nous observons une convergence de ces systèmes vers les systèmes embarqués, dont l'architecture manycoeur est le parfait exemple.

2 Caractéristiques des architectures sur puce

2.1 Infrastructure de communication sur puce

Dans les systèmes monopuce intégrant plusieurs unités de calcul, il est important de pouvoir augmenter le nombre de coeurs sur la puce, mais aussi d'assurer une meilleure communication entre eux. En effet, la question sur la communication sur puce contraint le passage à l'échelle en terme de latence et bande passante. Il est donc important d'offrir une large bande passante tout en limitant la consommation en énergie et la dissipation thermique. Les topologies usuellement employées sont celles provenant de la culture des réseaux : anneau (a), grille (b), crossbar (c), tore (d), hypercube (e), arbre (f). Mais les spécificités propres aux réseaux sur puce posent des contraintes dans le choix de la topologie (voir figure 1) : Dans le monde industriel, la principale topologie employée à ce jour sur les puces homogènes est la grille 2D (b). Les raisons généralement invoquées pour justifier ce choix sont la simplicitè de l'architecture du réseau et la simplicité du routage. l'architecture crossbar (c) est caractérisée

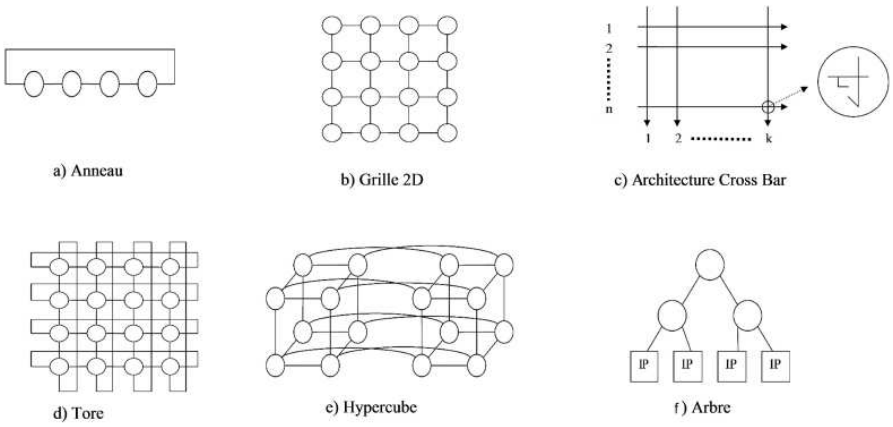

FIGURE 1 – Différentes topologies des réseaux sur puce

par une connexion matricielle. Elle permet de connecter n*k unités de calcul. Ce type d'architecture est particulièrement utilisé pour les réseaux dynamiques. Aujourd'hui, pour les systèmes manycoeurs la diminution de la période d'horloge et de la finesse de gravure empêchent la réalisation de circuits totalement synchrones ; on parle alors de circuits localement synchrones et globalement asynchrones (Globally Asynchronous, Locally Synchronous : GALS). Le circuit est alors organisè en îlots de synchronisme, et ces îlots sont connectés par un réseau sur puce (Network on Chip : NoC). Les réseaux sur puce peuvent apporter une solution aux problèmes causèes par les limitations des bus. Un réseau sur puce est une fabrique de communication multi-hop avec un système de commutation de paquet. Les techniques de commutation permettent l'optimisation de l'occupation des ressources (mémoire et canal de sortie) lors de transmission d'un paquet. En effet, les ressources utilisées pour le chemin complet ne sont pas réservées durant tout l'envoi, elles le sont juste lors du passage des paquets. Différents modes de commutation sont proposés (Store-and-Forward et Cut-Through). N'étant pas le sujet de ce travail, nous ne développons pas ce point.

Finalement, dans un réseau sur puce les composants sont connectés à travers des connexions Point à Point à l'aide d'une interface réseau. Actuellement, la tendance est d'utiliser des circuits intégrés en 3D comme solution permettant de contourner la contrainte liée à la surface de gravure de la puce d'un côté. D'autre part des liaisons photoniques sont utilisées pour véhiculer l'information entre les noeuds, afin de palier au souci du ralentissement des communications dans les puces denses (ie. nombre important d'intermédiaires entre deux noeuds).

2.2 Paradigme structural des mémoires cache

Aussi bien que n'importe quel élément d'un système, la mémoire présente différents types de technologies, d'organisations, de performances et de coût. Elle est d'une grande importance dans le cycle de conception des futurs systèmes. Notamment, dans la conception des manycoeurs. Le problème de la quantité de données est sans limite : si la capacité existe, les applications seront développées pour l'exploiter. Il est, en un sens, plus facile d'aborder la question de la vitesse d'un processeur. Or, pour obtenir de meilleures performances, la mémoire doit pouvoir suivre le rythme du processeur. L'unité de calcul ne doit pas attendre les instructions ou opérandes dont il a besoin pour exécuter des instructions. Un compromis doit être fait entre ces trois caractéristiques : coût, capacité et temps d'accès. À tout

moment, dans la gamme de technologies qui implémentent les systèmes à mémoire, le concepteur tient compte des relations suivantes :

-Plus le temps d'accès est court, plus le prix par bit est élevé ;

-Plus la capacité est importante, plus le prix par bit est faible ;

-Plus la capacité est importante, plus le temps d'accès est long.

Le concepteur est confronté à un dilemme ; il voudrait exploiter les technologies qui proposent une capacité de mémoire importante, d'une part parce qu'il a besoin de cette capacité et d'autre part en raison du faible coût par bit. En revanche, pour répondre aux besoins en matière de performance, le concepteur doit faire appel à des mémoires plus chères, de capacité moins importante et à des temps d'accès plus courts. La solution consiste à ne pas se limiter à un composant ou à une technologie unique, mais à employer une hiérarchie mémoire (voir figure 2). Voici ce que l'on rencontre à mesure que l'on descend dans cette hiérarchie :

-Baisse du coût par bit ;

-Augmentation de la capacité ;

-Baisse de la fréquence d'accès du processeur à la mémoire.

FIGURE 2 – Structure de mémoire hiérarchique

Dans le cadre de ce stage, nous nous intéressons particulièrement aux mémoire caches. En effet, dans une structure de systèmes répartis où plusieurs composants (typiquement un processeur) possèdent un cache et où la mémoire principale est partagée, un nouveau problème se présente. Si on modifie les données de l'un des caches, on n'invalide pas uniquement le mot correspondant dans la mémoire principale, mais également ce mot dans les autres caches. Même avec une stratégie d'écriture simultanée, les autres caches peuvent contenir des données invalides. Pour remédier à ce phénomène, on fait appel à un système maintenant la cohérence des caches, dont il existe plusieurs approches (traitées plus loin dans le rapport). Pour des architectures où plusieurs coeurs cohabitent sur une même puce, et partagent la mémoire externe, des supports efficaces sont intégrés pour assurer la cohérence des caches. Au cours des dernières décennies, les nouvelles contraintes technologiques, entraînées par l'augmentation du nombre de coeurs intégrés sur puce, constituent un grand défi. Elles exigent de nouvelles solutions pour un faciliter le passage à l'échelle. A ce niveau, de nombreux protocoles de cohérence ont été proposés, et ils ont tous les mêmes buts : la performance (latence, temps d'accès..), le trafic interne et la surface (silicium) de gravure. L'ensemble d'une architecture est conçu comme un réseau point à point

de coeurs de calcul ou les différents coeurs sont liés par un support partagé (Bus, Crossbar), comme par exemple le Dual-core IBM Power6 (Le et al., 2007) et le 8-core Sun UltraSPARC T2 (Shah et al., 2007). Chaque coeur comporte une unité de traitement à un ou plusieurs niveaux de cache , et une interface réseau. Dans notre travail, nous considérons des noeuds à 2 niveaux de caches. Le premier niveau est privé et uniquement accessible par le noeud père. Le second est physiquement distribué, mais peut être logiquement partagé (voir figure 3). Ce type d'architecture fournit des performances considérables, mais avec le souci de passage à l'échelle des manycoeurs des améliorations seront envisageables. La dualité sur l'aspect privé ou partagé du dernier niveau de cache , a donné naissance à plusieurs propositions en essayant de trouver le meilleur compromis entre ces deux configurations. Ceci est donc sujet de plusieurs travaux de recherche dans le domaine.

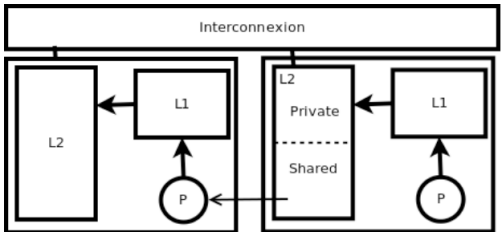

FIGURE 3 – Structure à cache L2 logiquement partagé

La majorité des propositions optent pour le choix du cache L2 partagé et allouent de l'espace suivant les exigences de chaque noeud. Dans ce cas les données ne se retrouvent pas forcément à proximité de leurs noeuds demandeurs. Raison pour laquelle ces propositions génèrent une latence élevée et un trafic interne important.

Ensuite, d'autres propositions sont des améliorations de ce même modèle. Elles essaient de réduire le temps d'accès aux données tout en gardant la proximité physique au noeud demandeur. Ce modèle connu sous le nom de Spilling ou N-Chance Forwarding [1], envisage de partager dynamiquement le cache L2 suivant les besoins de l'application en cours. Il consiste à transférer les blocs de cache éjectés vers d'autres noeuds de manière arbitraire. Le choix de la destination de manière arbitraire, ne peut être la meilleure solution dans certains cas (exp : un grand nombre de noeuds). Cela peut même engendrer une perte d'énergie et une augmentation inutile du trafic sur puce. Pour améliorer ces points, d'autres propositions ont été faites, par exemple les techniques dites Distance-Aware spilling [1] et Selective spilling [1] développées par Intel Barcelona. Ces techniques permettent d'éjecter les blocs fréquemment utilisés vers des noeuds voisins et décident en fonction de l'éventuelle réutilisation de chaque bloc s'il s'avère nécessaire de les garder en cache ou pas. Les prochaines générations de micro-architectures risquent d'être limitées par le taux d'accès en mémoire externe (Off-chip misses), mais aussi par la charge du réseau intra-puce (On-chip traffic). Dans cette perspective, le groupe Larabee de Barcelone a proposé une approche inovatrice, basée sur le principe de cache coopératif mais avec une meilleure capacité à s'adapter au type d'application. Cette approche appelée Cache Elastique [2], est perfectible notamment sur le choix de la meilleure destination, mais aussi sur l'efficacité de la gestion du partage du cache. Des grappes de serveurs aux manycoeurs, plusieurs modèles et protocoles on été utilisés pour gérer la cohérence de manière consistante. Dans ce rapport, nous allons présenter de près ces différentes méthodes qui visent à assurer une cohérence performante, dynamique et mieux adaptée aux exigences des futurs systèmes.

3 Notion de la cohérence de données

3.1 Définition de la problématique de cohérence

La mémoire cache est destinée à fournir une vitesse approchant celle des mémoires les plus rapides tout en offrant une taille importante à un prix inférieur de celui des mémoires à semi-conducteurs. Lorsque le processeur tente de lire un mot en mémoire, il vérifie d'abord s'il se trouve dans le cache. L'idée est que le processeur puisse récupérer des données fréquemment utilisées de manière rapide et efficace. Dans une structure où plusieurs composants possèdent une hiérarchie de cache et où la mémoire principale est partagée, le problème suivant se présente. Si on modifie les données de l'un des caches, on n'invalide pas uniquement le mot correspondant dans la mémoire principale, mais également ce mot dans les autres caches (si l'un d'eux contient le même mot). Aussi, une même donnée peut être présente à différents niveaux de la mémoire (cache , mémoire principale), ce qui explique le problème de cohérence et de propagation des modifications entre les différents niveaux. Dans une architecture multicoeurs, tout processeur modifiant la valeur du bloc dans sa mémoire privée fait que les autres copies du bloc (Mémoire principale, Caches des autres processeurs) ne représentent pas la valeur la plus récente de celui ci d'où les problèmes d'incohérence. Le schéma suivant est une illustration de ce phénomène sur plate-forme à deux coeurs (voir figure 4).

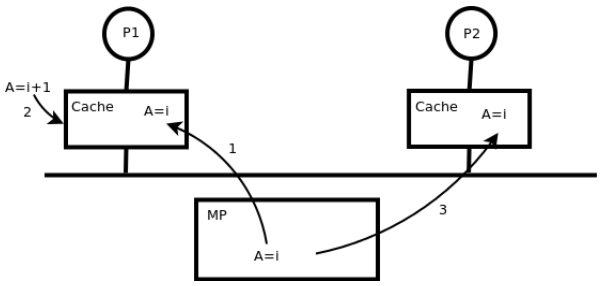

FIGURE 4 – Illustration du problème d'incohérence de caches

Le processeur P1 lit la donnée A de la mémoire principale (1). Il modifie son contenu (2). Le processeur P2 accède en lecture a cette même donnée dans la mémoire principale (3). Il n'a donc pas la dernière mise à jour de A, car le processeur P1 est le seul à voir sa propre modification de la donnée sur son cache. Un système est dit à cache cohérent si à tout accès à une donnée, la valeur retournée est celle résultant de la dernière opération d'écriture dans cette donnée. De ce fait, il existe plusieurs approches pour résoudre le problème de la cohérence de données. Ces approches dépendent de plusieurs critères.

3.2 Modèles de cohérence de cache

Le problème de la cohérence de données a été traité à plusieurs reprises, pour différentes architectures et différents type de systèmes. Pour des programmes séquentiels, s'exécutant sur des machines de type Von Neumann, le modèle d'exécution est naturellement clair. Les instructions sont exécutées dans le même ordre défini par le programmeur ou le compilateur. Cependant, avec l'utilisation des systèmes parallèles, notamment les multicoeurs, les opérations sur les données se font dans différents ordres. La question est donc de pouvoir assurer qu'au moment de chargement d'une donnée, le système récupère la dernière valeur écrite sur la mémoire. Le modèle de cohérence de données spécifie comment un noeud doit observer les accès mémoire effectués par les autres noeuds. Les incohérences peuvent être très

problématiques dans certains cas. Par contre , il peut arriver que l'on n'ait pas besoin d'une cohérence parfaite. On peut prendre pour exemple une application qui surveillerait la charge des processeurs d'un réseau d'ordinateurs pour que l'on puisse choisir sur quelle machine exécuter une tâche : les données qui représentent la charge des processeurs évoluent sans arrêt, il est donc inutile d'avoir toujours la dernière valeur exacte. Or, au moment d'une opération de lecture la question sur la dernière valeur valide reste généralement posée. Conséquemment, il reste nécessaire d'établir un modèle de cohérence pour répondre à cette question. Deux approches ont été considérées pour résoudre le problème de la cohérence :

L'approche de cohérence faible Weak Consistency [3] utilise la connaissance des opérations de synchronisation pour assouplir la cohérence de la mémoire. Par exemple quand on utilise un verrou pour une section critique, le système assure que les autres processus n'ont pas accès à ces valeurs pendant ce temps là. Les mises à jour ne sont propagées une fois la section critique terminée.

L'approche de cohérence forte est la plus utilisée. Cette approche assure une meilleure visibilité (localisation transparence de données). Elle met également des contraintes importantes sur l'ordonnancement des opérations (lecture/ écriture). Nous citons l'exemple du modèle de cohérence séquentielle, qui est basée sur deux hypothèses :

Toutes les lectures et écritures sont faites dans l'ordre du programme, comme les programmeurs s'y attendaient. Les opérations sur la mémoire venant de processus différents ont lieu dans un ordre séquentiel.

3.3 Protocoles de cohérence de cache

L'implémentation des différents niveaux de caches a un impact sur les modèles de cohérence ; problématique traitée plus haut (paragraphe 3.a). La cohérence de données entre les différents caches, nécessite un mécanisme de propagation de mises à jour qui soit efficace. Intuitivement, une technique implémentée dite write update [4], consiste à propager la valeur modifiée après toute opération d'écriture. Tous les processeurs ont donc en permanence des données à jour, ce qui à l'avantage de rendre les lectures très peu coûteuses. Le gros inconvénient de cette méthode en revanche est que les propagations des actualisations doivent être totalement ordonnées pour respecter l'ordre des émissions de tels messages, ce qui augmente le coût.

Le fait que chaque processeur ait son propre cache où il peut modifier ses propres copies de données, introduit la notion de permission. En effet, un processeur ne peut écrire dans une donnée avant d'avoir la permission de tous les autres qui ont une copie de cette donnée. Ce modèle rassemble une famille de protocoles nommé write invalidate [4]. L'approche à invalidation permet plusieurs accès en lecture seule simultanés sur une donnée, mais un seul en écriture est possible. Lorsqu'un processus veut écrire sur une donnée, il doit envoyer un message aux autres processeurs pour invalider cette donnée et attendre leurs retours. Les mises à jour sont propagées uniquement quand un processus accède à une donnée invalide, ce qui évite les transmissions superflues. De plus le coût est réduit car les propagations n'ont pas besoin d'être totalement ordonnées. Le protocole à état appelé MESI, ainsi que ses différentes variantes, est le plus commun de cette famille. L'état de cohérence d'un bloc est stocké dans un tag à deux bits (voir figure 5).

Ce tableau (voir figure 6) présente les différents états d'un bloc et les permissions associées à chaque état pour un protocole MESI simple. Le cas critique est l'état Partagé : une écriture dans une ligne à l'état Partagé implique l'invalidation de toutes les autres lignes partageant cet état, donc une propagation des invalidations. La fourniture du contenu d'une ligne à d'autres caches est également plus complexe. C'est pour cette raison que des extensions du protocole MESI ont été définies et sont

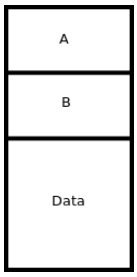

FIGURE 5 – Bloc de donnée cache

Modified	Exculsive	Shared	Invalid
AB=11	AB=01	AB=10	AB=00
Le contenu du bloc dans le cache est différent de celui de la mémoire principale.	Le contenu du bloc dans le cache est le même que dans la memoire principale.	Le bloc peut se trouver dans plusieurs caches.	Le bloc dans le cache ne contient pas de données valides.
Lecture/ Ecriture		Lecture	Aucune

FIGURE 6 – Le protocole MESI et les permissions d'accès par bloc

utilisées dans les multicoeurs, notamment MOESI (O pour Owner) et MESIF (F pour Forward). Pour le premier, l'état Owner signifie que la ligne de cache contient la donnée la plus récente qui peut se trouver aussi dans la mémoire cache d'un autre processeur. En revanche, la valeur contenue par la mémoire principale est incorrecte. Pour la deuxième variante de cette famille de protocoles, l'état Forward permet à un noeud de récupérer la valeur valide d'une donnée sans avoir à aller interroger la mémoire principale. Ceci conduit à une approche de cache coopérative, et réduit les accès en mémoire externe qui pénalisent les performances du système.

Troisième partie

Mécanismes de cohérence de données pour les architectures grande échelle

4 Gestion de la cohérence dans les systèmes répartis

Un système distribué est une collection de processeurs qui échangent des messages via un système de communication constitué de connexions de logiciels. Il est destiné à des applications distribuées. Différentes tailles de systèmes distribués existent, ce qui nécessite un passage à l'échelle flexible. Il existe plusieurs techniques permettant d'assurer le partage des données dans les architectures distribuées. Notamment pour les systÃ¨mes de partage de fichiers [5] et les applications du web qui est un bon exemple de partage de fichiers à grande échelle [6].

4.1 Systèmes à mémoire virtuellement partagée

Un Systèmes à Mémoire Virtuellement Partagée (MVP) est un système dans lequel le programmeur a l'impression que l'ensemble des mémoires distribuées n'en forment qu'une seule. Son intérêt est de permettre l'utilisation d'un modèle de programmation qui a des avantages par rapport aux modèles basés sur l'échange de messages. Les processus accèdent à la MVP par des lectures et des mises à jour sur ce qui leur semble être de la mémoire ordinaire à l'intérieur de leur espace d'adressage. Mais un système tournant en tâche de fond assure de manière transparente que les processus s'exécutant sur différents ordinateurs observent les mises à jour effectuées par d'autres processus. Les données sont donc récupérées quelque soit leur localisation physique. La transparence des données permet de garder un modèle de programmation classique par mémoire partagée. Néanmoins, tout échange de données entre mémoires distribuées reste masqué. Lorsqu'on accède à une donnée partagée, la MVP doit garantir que cette donnée est à jour, compte tenu du modèle de cohérence employé. Elle doit pour cela localiser le noeud stockant la dernière valeur de la donnée. En conséquence, les systèmes de MVP doivent stocker des informations à propos de chaque ligne mémoire. Les premiers systèmes à MVP tel Ivy [7] utilisaient le modèle de cohérence forte. Un modèle de cohérence dit d'entrée est apparu dans le système Midway [8] dans lequel la cohérence d'entrée requiert que chaque variable partagée soit explicitement associée avec un verrou de synchronisation. Lors de l'acquisition d'un verrou, toutes les variables associées à ce verrou sont mises à jour.

Avec l'augmentation du besoin en calcul des architectures de type fédération de grappes ou grappes de grappes sont exploitées. Pour cela les systèmes à MVP doivent prendre en compte les temps de latence du réseau entre les différents noeuds. D'autres protocoles de cohérence tenant compte de cette contrainte apparaissent. Nous citons le protocole dit Clustered Lazy Release Consistency [9, 10] qui propose de limiter les communications inter grappes. Chaque grappe désigne un noeud pour servir de cache pour tous les autres.

Si les MVP ont permis la mise en oeuvre de mécanismes efficaces de partage de données avec des limites de passage à l'échelle, d'autres types de systèmes répartis se sont focalisés sur la gestion des données à plus grande échelle à savoir les systèmes Pair à Pair.

4.2 Systèmes Pair à Pair

Un système Pair à Pair est un système distribué où l'on distingue pas entre les rôles des noeuds (client/ serveur). Le premier système Pair à Pair apparu en 1999, est Napster où chaque noeud est à la fois producteur et consommateur de fichier MP3. Grâce à l'évolution d'internet et l'augmentation du débit de transfert, plusieurs applications de partage de fichiers de type Pair à Pair sont nées, intégrant de plus en plus de noeuds actifs (edonkey, emule). Le principe des systèmes Pair à Pair est de manipuler des données non modifiables. Il est donc pas autorisé d'utiliser des processus d'écriture d'où pas de problème de cohérence. Or, certains systèmes de gestion des fichiers de ce type donnent la possibilité de modifier des données et donc sont dotés de mécanismes de cohérence des données. Le principe général de la cohérence dans ce cas est d'insérer un nouveau bloc à chaque fois qu'un ancien est modifié. De nombreux mécanismes de coérences sont proposés [11, 12, 13, 14, 15], nous pouvons citer : OceanStore [12] est un exemple d'une application de stockage Pair à Pair qui utilise un modèle de cohérence hiérarchique tolérant aux fautes. C'est un protocol basé sur la rendodance de données. Il assure deux niveaux de garantie pour deux types de copies dans le système : les copies primaires ayant la version la plus récentes de mise à jours et les copies secondaires repésentant des versions antérieures. Pastis [16] est une structure de données fortement inspirée du système de fichier Unix dont la cohérence de données est assurée selon deux modèles :

- Close-To-Open : les modifications ne sont prises en compte qu'après la fermeture du fichier ;
- Read-Your-Writes : chaque site ne voit que ses propres modifications. En cas de conflit, le dernier écrivain voit ses modifications prises en compte.

.

Contrairement aux systèmes à MVP, les informations sur la localisation des données et leur état de cohérence évoluent en fonction de la dynamique du réseau dans les systèmes Pair à Pair. Ce qui fait la force de ces derniers par rapport aux premiers.

4.3 Systèmes sans fils Ad Hoc

Les noeuds mobiles communiquent entre eux de manière décentralisée. Et les données sont accessibles depuis leurs noeuds sources à travers un environnement à multiple sauts. Chaque noeud mobile conserve la donnée accédée dans son propre cache , pour une future utilisation éventuellement de la part d'un noeud voisin [17] . La disponibilité des données et leurs accessibilité constituent le plus grand défi de ces systèmes. La coopération des caches permet d'améliorer la disponibilité des données en mettant l'accent sur la coordination entre les noeuds mobiles. Ceci a reçu une bonne concentration de nombreuses recherches, ce qui à conduit à plusieurs stratégies [18, 19, 5]. Nous traitons quelques unes des plus récentes dans ce qui suit :

-Le modèle Zone Cooperative Caching :
Une zone encadrant un noeud mobile est constituée par tous les noeuds qui en sont à un pas. Le principe de cette stratégie est de renforcer la coordination entre voisins proches. La recherche d'une donnée est donc faite au niveau locale, ensuite sur la zone de coopération et enfin dans tous les noeuds qui se trouvent sur le chemin vers le serveur (noeud source). En effet, les données ne sont modifiées qu'au niveau des serveurs. Et chaque noeud est chargé de satisfaire non seulement ses requêtes locales mais aussi toute requête le traversant pour aller au noeud source. il s'agit d'une méthode à invalidation. Si la donnée est mise à jour, toutes ses copies caches sont invalidées. Si le serveur fait partie d'un voisinage, on ne met pas les données dans le cache local. Les noeuds mobiles d'une même zone stockent des données différentes ce qui aide à réduire la consommation d'énergie et de la bande passante. S'il

n'y a plus assez d'espace de stockage dans le cache , on remplace le plus vieux bloc. Or, la latence peut devenir importante si la donnée n'existe pas chez les voisins des noeuds intermédiaires, car chacun de ces noeuds doit attendre une réponse de son voisinage avant de transférer la requête vers le serveur.

-Le modèle Group Caching :

Sur une plage de transmission, chaque noeud envoie périodiquement des messages Hello à ses voisins proches et forme avec eux un Groupe. Les groupes ont chacun un noeud maître qui maintient la communication avec tous les noeuds membres. Le noeud maître vérifie l'état des caches du groupe par envoi des messages de contrôle. Tout élément du groupe possède une table où il enregistre l'état de son propre cache et celui des autres noeuds.

À la réception d'une requête le noeud vérifie dans cette table l'existence de la donnée dans son cache local ou dans ceux des autres membres du groupe. Dans le cas d'un défaut de cache la requête est redirigée au prochain noeud sur le chemin vers le serveur. En cas de saturation du cache local d'un noeud, il existe deux approches. Une première consiste à stocker chez un voisin proche (celui ayant de la mémoire disponible). La deuxième approche traite le cas où aucun noeud membre n'a suffisamment d'espace pour héberger la donnée reçue. Le noeud en réception vérifie dans sa table si aucun des noeuds n'est en possession de cette donnée. Si oui il n'est pas nécessaire de les mettre en cache. Dans le cas contraire, on cherche le noeud qui comporte le plus vieux bloc, et on le fait remplacer par la nouvelle donnée. Si un noeud mobile ne reçoit pas de message Hello de son voisin pendant un certain nombre de cycles, il considère que ce voisin est déconnecté. Et puis il met à jour sa propre table en supprimant les enregistrements du voisin déconnecté. La nécessité d'envoyer fréquemment des messages de connexion à son voisinage, engendre une grande consommation d'énergie ce qui représente le point faible de cette stratégie.

5 Techniques de cohérence pour les systèmes à multiprocesseurs

Les multiprocesseurs sont des architectures parallèles avec plusieurs processeurs ayant chacun sa hiérarchie de caches et une mémoire principale logiquement partagée. Le même espace d'adressage est donc visible par tous les processeurs. Cette mémoire logiquement partagée peut être également être physiquement partagée ou distribuée. Le problème de la cohérence tel qu'il est défini au chapitre 1 se complique avec l'existence des caches associés à chaque processeur. Pour gérer ce problème dans les multiprocesseurs, on fait appel à différents mécanismes [20, 21, 22], qui peuvent être centralisés ou distribués.

5.1 La famille de protocoles centralisés par répertoire

Le premier protocole opérationnel de cohérence de cache centralisé a été présenté par Tang en 1976 [23]. Une dernière version améliorée a été présentée par Censier et Feautrier en 1978 [24]. Gérer la cohérence dans un système à multiprocesseurs implique de connaître la relation entre chaque ligne en mémoire principale et toutes les copies de cette ligne dans les différents caches. Les requêtes de lecture et d'écriture effectuées par les nombreux processeurs engendrent différentes versions de ces copies. Dans les protocoles centralisés, la relation est établie par des liaisons entre chaque ligne de la mémoire principale et chaque copie possible dans les autres caches [25]. L'information permettant d'assurer la cohérence est centralisée au niveau de la mémoire dans un répertoire [26]. Le principe de base du protocole à répertoire est d'associer à chaque ligne de la mémoire principale un vecteur de K bits indiquant la présence ou non de la ligne correspondante dans chacun des K caches du multiprocesseur. Un bit modifié indique si la ligne de la mémoire principale est différente ou identique à toutes les lignes

correspondantes. En plus, dans chaque ligne des caches des processeurs, il y a un bit valide qui signifie que la cohérence est normale. Et un bit privé, qui indique que ce cache possède la seule copie valide de toutes les autres copies. Les principaux inconvénients du modèle à répertoire se résument aux deux points suivants :

-La taille du répertoire est proportionnelle au nombre de lignes de la mémoire principale et au nombre de processeurs. Cette taille varie donc comme MP*N, MP étant la capacité de la mémoire principale et N le nombre de processeurs. On voit clairement que ceci posera des problèmes de passage à l'échelle ;

-Le répertoire centralisé est une extension de la mémoire principale. Les opérations des processeurs impliquent des transactions entre les contrôleurs de cache des processeurs et le contrôleur de la mémoire principale. Ce qui peut être très pénalisant dans le cas d'une activité intense.

5.2 Le modèle décentralisé par espionnage

Chaque cache ayant une copie de la donnée d'un bloc de la mémoire physique a aussi une copie de l'état de partage du bloc. Aucune information centralisée n'est utilisée [27, 28]. Tous les contrôleurs de caches surveillent, ou espionnent le réseau les connectant à la mémoire pour savoir si une copie du bloc est en transfert sur le bus. Cela permet également des relations directes entre caches. La cohérence par bus est basée sur un certain nombre de principes. Les transactions du bus sont visibles par tous les processeurs, et notamment par les contrôleurs de cache. De part son aspect basé sur la diffusion, la technique d'espionnage nécessite une bande passante élevée. Tant que le réseau n'est pas saturé, la cohérence par espionnage fonctionne de manière satisfaisante. Pour un nombre important de coeurs, la cohérence par diffusion de données n'est pas une solution efficace.

Quatrième partie

Cohérence des caches dans les microarchitectures

Une large gamme de processeurs actuels est constitués de plusieurs dizaines de coeurs, dont Intel Xeon à 10 coeurs, l'AMD Opteron composé de 12 coeurs ou encore le Tilera TileGx qui atteint les 100 coeurs. Aujourd'hui, l'industrie se dirige vers la fabrication de systèmes à plusieurs centaines de coeurs tels Intel SCC (1000 coeurs), Adaptive (4096 coeurs) ou Kalray (1024 coeurs). Ce chapitre présente les structures de caches les plus utilisées dans ce domaine, ainsi que les méthodes et aspects actuels de la cohérence de données mises en place pour accompagner le développement de ces systèmes.

6 Structure à cache hiérarchique

6.1 Approche de caches partagés

Une des principales caractéristiques des architectures récentes, est qu'elles ont divers degrés de partage de mémoire aux différents niveaux de cache. La mémoire principale est généralement partagée par tous les coeurs. La plupart de ces architectures, ont des noyaux à cache L1 privé. Selon les besoins en application, le niveau L2 de cache est partagé par un ou plusieurs noeuds. Les hiérarchies de cache qui existent dans les processeurs disponibles actuellement n'implémentent qu'une partie de la structure hiérarchique classique. Le schéma présente les différentes hiérarchies de cache de processeurs multicoeurs (voir figure 7).

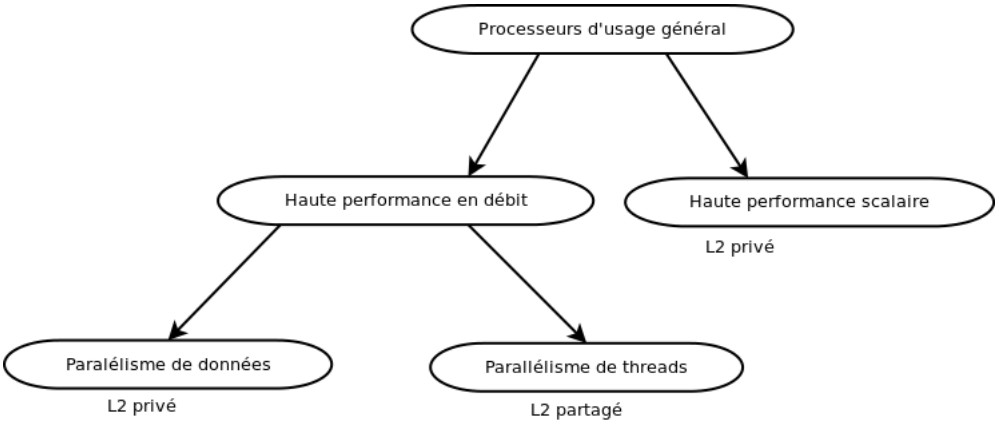

FIGURE 7 – Hiérarchie de cache de processeurs multicoeurs telle que présentée par Intel

Si le principe de cache L2 partagé représente la tendance des architectures nouvelles, c'est justement grâce à sa pertinence dans l'amélioration des performances au niveau de la communication entre les différents coeurs. Or, ce modèle engendre des problèmes liés aux mécanismes de la cohérence de données lors du passage à l'échelle. Le tableau suivant résume les avantages et les inconvénients de la hiérarchie de cache avec et sans la propriété de partage du niveau L2 (voir figure 8).

6.2 Quelques modèles industriels sur le marché des microprocesseurs

Ce chapitre décrit les hiérarchies de mémoires utilisées dans quelques systèmes des plus récents dans le marché des microprocesseurs. Généralement, le premier niveau cache est privé, mais le cache

Cache L2 partagé	Cache L2 privé
+ : Minimisation du flux de communications avec la mémoire externe, + : Optimisation de l'utilisation de l'espace , + : Migration plus faciles des tâches. - : Consommation de la bande passante, - : Contention de la mémoire , - : problème de cohérence .	+ : Pas de problème de contention - : Communication/migration plus coûteuse, passage systématique par Le cache local

FIGURE 8 – Cache L2 privé versus partagé

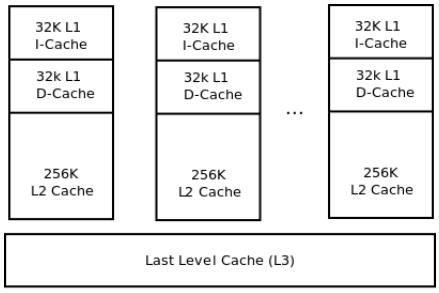

FIGURE 9 – Configuration du cache Nehalem

L2 diffère d'une architecture à une autre. Le paragraphe (a), présente la génération Nehalem d'Intel. Dans le paragraphe (b), il est décrit le microprocesseur d'AMD Opteron. Ensuite, le paragraphe (c) présente le multicoeur sur Ultrasparc T2. Le Cortex A9 d'ARM est présenté dans le dernier paragraphe (d).

-**Le processeur Nehalem Intel** : Le 27 juillet 2006, Intel lance ses processeurs basés sur la nouvelle architecture Core 2 Duo. Vers la fin de cette même année, le fondeur commercialise les premiers processeurs quadricoeurs. Plus tard en 2007, Intel annonce un grand saut dans le monde des architectures, avec une sortie des microprocesseurs appelés Nehalem prévu courant 2008. Cette architecture est considérée comme plus rapide au niveau des communications entre coeurs. Elle permet à travers, son organisation et sa gestion de mémoire cache d'améliorer efficacement les liaisons entre les multiples coeurs de la puce. En effet, l'architecture Nehalem est caractérisée par 3 niveaux de caches. Le niveau L1 est partagé en deux niveaux ; un cache L1 de 33 Kb pour instructions et un autre cache L1 de même taille pour les données. Le niveau L2 est de 256 Kb et il est unifié pour instructions et données. Le cache L3 représente la dernière amélioration par rapport à la famille Core 2 Duo. Il est d'une taille importante 8Mb. Contrairement aux deux premiers niveaux, ce niveau est partagé entre tous les coeurs du système (voir figure 9).

Avec ce protocole, seules les données en état Forward peuvent être dupliquées. L'état Forward est un état de partage où un noeud se permet de se procurer le droit exclusif de répondre aux requêtes d'une ligne de cache. Cette modification élimine donc le cas d'avoir de multiples positions caches pour répondre à une même requête sur une seule ligne de cache.

Le Nehalem a également apporté au niveaux interconnexions physiques entre grappe de processeurs. Avant l'architecture Nehalem, la connexion par bus système représentait un goulot d'étranglement à cause de la différence en vitesse entre le bus de communication et le processeur. L'architecture Nehalem, exploitant le schéma de communication point à point, facilite l'accélération des accès mémoire non

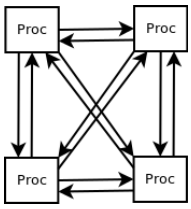

FIGURE 10 – Interconnexion QuickPath

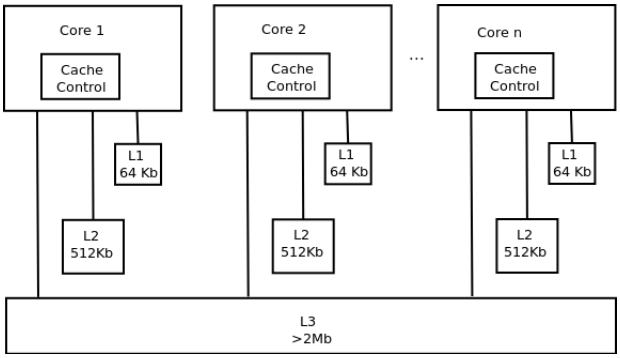

FIGURE 11 – La hiérarchie mémoire de l'AMD Opteron

uniformes. La communication est basée sur des liaisons courte dites QuickPath Interconnect (voir figure 10).

Intel a pu augmenter le nombre de défauts de cache L1 que son architecture est capable de traiter en parallèle. Cependant, au niveau de la nouvelle hiérarchie de cache L1, la bande passante du cache d'instructions n'a pas augmenté. Elle est toujours de 16 octets par cycle. Ce point pourrait s'avérer être une limitation dans une architecture orientée serveur. Le cache de données pour sa part voit sa latence augmenter à 4 cycles contre 3 sur le Conroe afin de permettre une meilleure montée en fréquence.

-L'architecture AMD Opteron : L'architecture AMD Opteron est similaire à celle d'Intel traitée précédemment (a). Elle est aussi basée sur 3 niveaux de cache. Le niveau L3 est également partagé. Or, ce dernier n'est plus inclusif ; il repose plus sur la duplication de données (voir figure 11).

L'AMD Opteron utilise également une version modifiée du protocole MESI. Le nouveau protocole proposé ajoute un état dit Owner ; ainsi cette notion permet à un noeud d'envoyer une donnée à un noeud demandeur sans avoir à passer par la mémoire principale, qui représente un coût d'accès suplémentaire. Ce qui implique que la mémoire principale n'a pas toujours la valeur la plus récente. Ceci élimine le fait de devoir envoyer toutes les modifications vers la mémoire principale a chaque fois qu'une donnée est appelée par un noeud. Le principe du protocole MESI est bénéfique si la bande passante entre processeurs est supérieure à celle entre processeur et mémoire principale, comme c'est le cas pour les microprocesseurs modernes.

Au niveau des communications, l'architecture Opteron intègre le bus HyperTransport qui permet d'améliorer les liaisons avec les autres périphériques (Off-chip). Cette technique augmente l'immunité au bruit et à la congestion, en implémentant d'un autre côté le protocole de transmission par paquet. Ce type de processeur a fait preuve de plus de fiabilité et de flexibilié au passage à l'échelle (16 coeurs), de finesse de gravure sur puce (33 nanomètres) et de faible consommation d'énergie. Plusieurs séries de cette architecture ont vu le jour dont l'Opteron 3000 sorti en début 2012, destiné à des usages pour

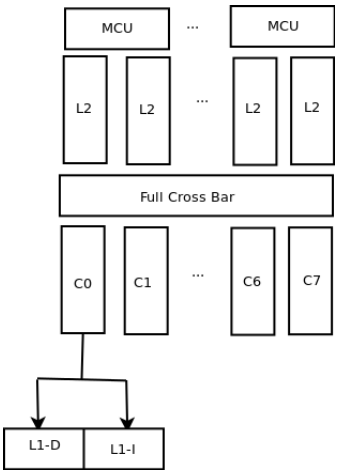

FIGURE 12 – Organisation du processeur Sun

serveurs d'entreprise fortement chargés.

-Le processeur Ultrasparc T2 de Sun : L'architecture Ultrasparc T2 de Sun, est un exemple de processeurs multicoeurs destinés aux serveurs d'infrastructures réseau. Elle représente un degré élevé de parallélisme. Le processeur est constitué de 8 coeurs tournant chacun à 1.4GHz. La hiérarchie de cache comporte un niveau L1 privé et non uniforme. Le cache L1 de données est d'une taille de 8Kb. Le cache d'instructions est d'une taille de 16 Kb. Le cache L2 est d'une taille de 4Mb. L'ensemble de caches L2 sont reliés à 4 contrôleurs de mémoire (MCU).

Des connexions en matrice sont établies de manière à assurer la communication de tous les processeurs avec tous les caches et vis-versa. Le cache L2 est divisé en 8 banks où des connexions sont assurées entre tous les caches L1 et ces derniers (voir figure 12).

-Le processeur Cortex A9 MP Core d'ARM : Le processeur Cortex A9 MP Core d'ARM est un multicore conçu pour les contextes embarqués. Il est très flexible et s'adapte à différents types d'applications. Pour utilisation dans des appareils mobiles, la consommation du processeur est extrêmement réduite. Il est constitué de 4 coeurs tournant à une vitesse allant de 800Mhz à 2Ghz. Il est donc possible de choisir entre la version qui considère la contrainte de consommation d'énergie et celle qui choisit les performances (vitesse de traitement). Une unité de contrôle par espionnage de bus implémente le protocole MESI amélioré. Le choix du niveau de cache L2 est optionnel, et peut être externe à la puce (voir figure 13).

7 État de l'art des protocoles de cohérence pour les multicoeurs

La multiplication du nombre de processeurs sur puce, l'évolution de la charge de travail ainsi que l'augmentation continue du coût d'accès en mémoire externe par rapport au coût d'accès en cache sur puce, forment un ensemble de facteurs qui apportent beaucoup d'intérêt à l'organisation de l'espace de stockage sur puce (ie. Caches hiérarchiques). Cette organisation a pour objectif d'arriver à satisfaire le maximum de requêtes sur puce, sans avoir à être pénalisé par des accès multiples à la mémoire externe. En plus, afin de réduire la latence d'accès aux données sur puce, il est nécessaire qu'elles soient stockées à proximité physique de son noeud demandeur. Dans une structure à cache hiérarchique, le cache L2

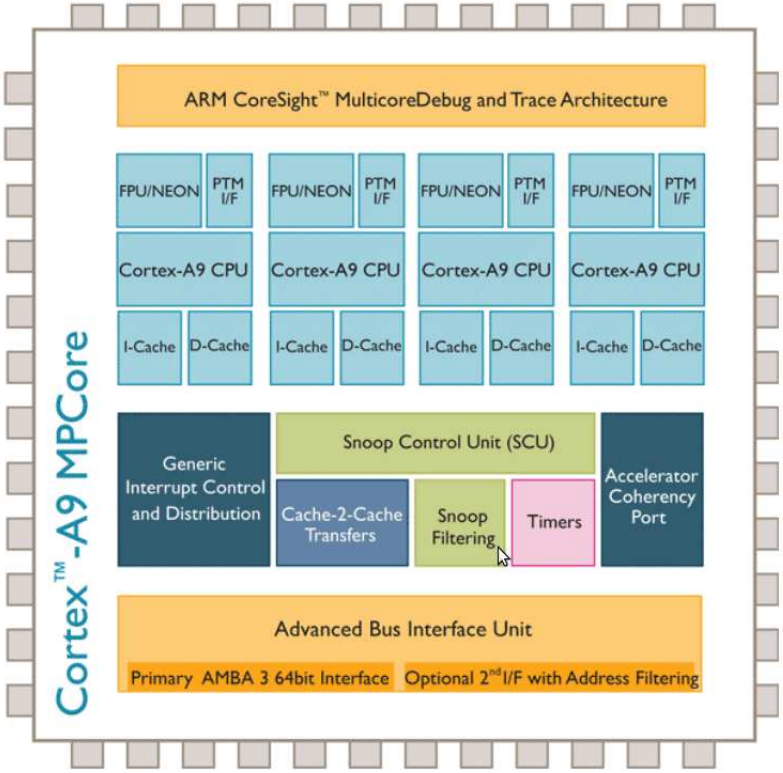

FIGURE 13 – Architecture ARM Cortex A9 MPCore

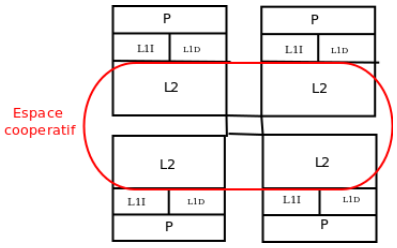

FIGURE 14 – Zone coopérative de 4 coeurs

peut être privé ou partagé. Comme vu précédemment, l'aspect partagé permet de réduire le nombre de défauts de cache , tandis qu'un cache privé réduit la latence.
Ainsi, la technique de partage de cache entre différents noeuds permet d'adapter l'utilisation du cache au besoin de chaque noeud. Cette technique basée sur un protocole de coopération s'avère être un bon compromis entre les deux aspects (privé/ partagé).
Cette section étudie les avantages et limitations, ainsi que les évolutions de cette technique.

7.1 Protocole à cache coopératif

Le protocole de cache coopératif consiste à combiner les avantages des systèmes à caches privés et ceux à caches partagés [29]. Le principe de coopération est de constituer un espace de stockage logiquement partagé, composé de l'agrégation d'un ensemble de caches privés. Les données globalement actives sont ainsi maintenues au niveau de l'espace commun pour limiter le nombre d'accès hors puce. Le principe des caches coopératifs est inspiré des algorithmes de coopération des caches de fichier ou des caches web. Étant physiquement séparés, ces caches se partagent leur espace libre afin de réduire le trafic d'accès au serveurs [30]. De manière similaire, pour des systèmes sur puce le protocole coopératif consiste à créer un grand espace commun, à partir des caches privés de l'ensemble des unités du système. Ceci permet d'adapter dynamiquement l'utilisation des ressources au besoin en espace de stockage des différentes applications (voir figure 14).

Le mécanisme de cache coopératif est mis en oeuvre à l'aide d'unités matérielles spécialisées dans le contrôle de l'allocation des données dans les caches. En effet, la gestion du partage de données stockées dans la zone coopérative, est effectuée par une unité de contrôle de cohérence de cache. Dans le modèle de cache coopératif proposé par Chang et Sohi [31], la gestion de la cohérence est centralisée. Elle est assurée par un répertoire commun CCE (Centralized Cooperative Engine) qui comporte la duplication de toutes les informations de cohérence des caches L1 et L2. En effet, en cas de défaut d'accès au cache local, le CCE redirige la requête vers le noeud en possession de la donnée et cette dernière est ensuite transférée de cache en cache jusqu'au noeud demandeur.

Lors de l'éviction d'une donnée du cache L2 locale, le CCE transfère la donnée sur un autre cache privé de la zone partagée. Afin de mieux utiliser l'espace de stockage commun, le contrôleur de cohérence utilise une politique de remplacement dite de N-Chance Forwarding [1], qui permet d'éviter qu'un bloc ne circule infiniment sur le réseau. La technique du N-Chance Forwarding consiste à associer un compteur à chaque bloc de donnée. Tout bloc est donc autorisé à migrer N fois avant d'être éjecté hors puce. Ce compteur est initialisé à chaque fois que la donnée est nouvellement chargée sur la puce.

Cependant, l'approche de cache coopératif à répertoire centralisé représente plusieurs limitations. La principale limitation rencontrée est la restriction liée au passage à l'échelle du répertoire centralisé. La deuxième limitation est liée à la consommation d'énergie au niveau de l'unité de contrôle centralisée

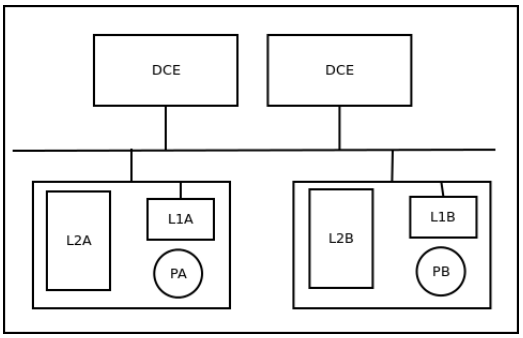

FIGURE 15 – Structure de cache coopératif à contrôle distribué

dont la charge en nombre de vérification par requête augmente avec l'augmentation du nombre de noeuds. Une nouvelle approche est proposée pour rémédier à ces limitations. Il s'agit d'une stratégie de gestion de cohérence de données distribuée appelée DCC (Distributed Coopérative Caching) [32]. La mécanisme du DCC est conçu principalement pour résoudre le problème de passage à l'échelle de l'ancienne configuration. Le principe de l'approche distribuée est de diviser l'unité de contrôle CCE en plusieurs unités distribuées DCC, dont chacune s'occupe d'une plage d'adresses. Les DCC comportent des informations sur la distribution des blocs sur les différents coeurs (Bloc To Home Node) (voir figure 15).

Nous pouvons aussi noter l'introduction de deux mécanismes permettant de limiter le nombre de destinations possibles lors de l'éjection d'un bloc Distance-Aware Spilling, et de restreindre le droit de migration sur d'autre caches aux blocs blocs prochainement utilisés Selective Spilling [1].

Le paragraphe suivant présente une nouvelle approche de hiérarchie mémoire qui permet une meilleure dynamique aux architectures sur puce.

7.2 Mécanisme de caches coopératifs élastiques

Le cache élastique Elastic Cooperative Caching [2] est introduit comme une extension du cache coopératif. Il offre un mécanisme de partage de cache L2 afin de répondre au mieux aux besoin des applications. Dans ce mécanisme, le partitionnement du cache L2 en zones privée et partagée est géré dynamiquement selon les accès aux données de chacun des noeuds. Les noeuds ayant une faible utilisation de leurs données locales permettent de créer un espace de stockage partagé plus large. Tandis que les noeuds qui accèdent fréquemment à leur cache local participent moins à l'espace de coopération.

Cette configuration permet d'assurer un comportement élastique, autonome et adaptatif du cache L2. Elle autorise également le passage à l'échelle via son aspect distribué. Le mécanisme de cache élastique tend à équilibrer l'utilisation de l'espace de stockage commun, en dépit de la différence en besoin mémoire des noeuds. Outre tous les mécanismes proposés au préalable, concernant le partage des ressources mémoire d'une structure à cache hiérarchique, l'aspect élastique des caches reste le seul qui considère le comportement dynamique des différentes applications. Pour concevoir une hiérarchie mémoire qui passe à l'échelle, la proposition du cache élastique utilise une structure distribuée pour la gestion de la cohérence, à savoir la DCE. L'allocation des blocs migrant se limite au voisinage proche. Une autre spécificité du mécanisme élastique par rapport au mécanisme distribué de base est que l'on ne peut pas déplacer un bloc de donnée plus d'une fois (1-Chance Forwarding). La structure des caches élastiques introduit deux unités de contrôle de caches L2.

```
If Private_LRU_Hit then
   Increase Counter
EndIf
If Shared_LRU_Hit then
   Decrease Counter
EndIf
If Repartition_Cycle then
   If Counter > Upper_Threshold then
      Add_Private_Way
      Send Repartition_Info_Msg
   ElseIf Counter < Lower_Threshold then
      Add_Shared_Way
      Send Repartition_Info_Msg
   EndIf
   Clear Counter
EndIf
```

FIGURE 16 – Algorithme de partitionnement de cache

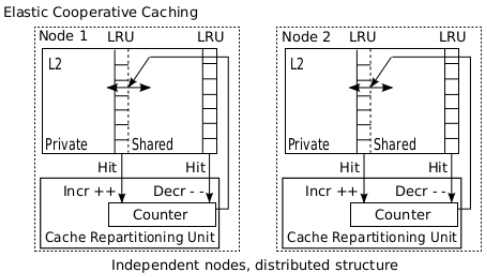

FIGURE 17 – Structure de l'unité de partitionnment de cache

-Unité de partitionnement : c'est la partie de contrôle qui assure le comportement adaptatif du partitionnement de cache L2 en deux zones (privé/partagé). Elle comporte un compteur d'accès qui s'incrémente à chaque succès d'accès au bloc LRU de la zone privée, et se décrémente s'il s'agit du LRU de la zone partagée. La mise à jour du partionnement se fait de manière cyclique. Tous les N nombre de cycles processeur, le partitionnement du cache est modifié, en fonction de la valeur du compteur d'accès, l'algorithme donné dans la figure 16 décrit le protocole de partitionnement.

En plus, lors de la mise à jour du cache , les blocs touchés par la propagation d'une zone, seront progressivement remplacés par les nouveaux blocs. Ceci représente un avantage en terme de limitation des défauts de cache. Cependant, comme le partitionnement est basé sur la fréquence d'accès au LRU, si la zone partagée n'est pas utilisée, la partie privée continue à augmenter même s'il n'est pas forcément nécessaire (voir figure 17).

Le partitionnement du cache étant effectué tous les N cycles processeur, implique la nécessité de choisir une valeur de N pertinente en fonction du comportement de l'application (ex. La fréquence d'accès mémoire). Nous pouvons nous demander s'il n'est pas plus pertinent de fixer les N nombre de cycles en fonction des actions du protocole plutôt qu'en fonction du temps.

-Unité d'allocation de blocs : l'allocation des blocs est un mécanisme qui permet de distribuer de manière efficace les blocs sur les noeuds destinataires. Les informations concernant le partitionnement du cache d'un noeud sont diffusées à tous les autres noeuds voisins, à chaque mise à jour. Ces informations sont ainsi utilisées pour décider du noeud destinataire de la prochaine éviction. L'unité d'allocation de blocs utilise un arbitre à tourniquet (Round Robin) avec un vecteur de bits que l'on met

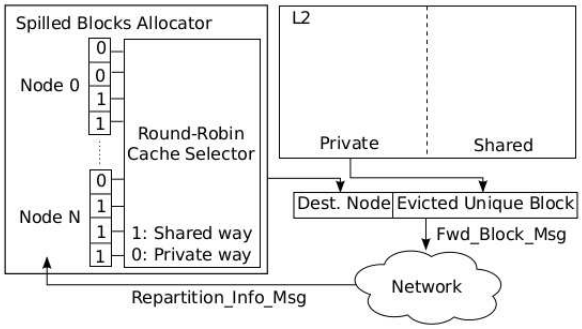

FIGURE 18 – Structure de l'unité d'allocation de blocs

à jour par les informations de partitionnement de chaque bloc dans chaque cache. En effet, lorsque l'on déplace un bloc de la partition privée d'un cache local l'arbitre choisit comme destinataire le noeud du bloc partagé suivant. De cette manière la distribution de la migration des blocs dans la zone agrégée se fait de façon équitable entre les noeuds d'un voisinage (voir figure 18).

Si le noeud utilise moins fréquemment son cache local, le protocole de partitionnement va continuer à charger des données dans la zone partagée, sans toutefois avoir besoin de les réutiliser. Pour remédier à cette limitation, on introduit un mécanisme adaptatif d'éjection permettant ainsi d'interdire la migration des blocs qui ne seront plus réutilisés.

Dans le prochain paragraphe, nous étudions les limitations de performances des mécanismes utilisés actuellement pour le partage élastique de cache L2 en cas de hausse de charge de travail dans un voisinage coopératif. Ce qui nous permis de proposer des améliorations à ce propos.

7.3 Limitations du modèle coopératif élastique

Le protocole de partitionnement de cache élastique est basé sur un seul compteur prenant en compte les accès réussis aux deux zones de la mémoire. Dans le cas d'un fonctionnement déséquilibré entre le noeud local et son voisinage, cette technique se montre efficace, car elle permet de partager l'espace de stockage selon le besoin en mémoire de chaque partie.Or, si l'ensemble d'un voisinage se voit fortement stressé par les accès à la zone privée et partagée, le compteur aura deux comportements distincts suivant le scénario d'accès en mémoire :

-Stagnation dans une valeur stable : c'est le cas d'une succession d'accès alternés entre cache privé et partagé. Conséquemment, le cache va garder son partitionnement courant. Ce qui sera traduit par une limitation de l'aspect adaptatif du mécanisme ;
-Forte oscillation : le compteur va osciller entre les valeurs limites définies par l'unité de partitionnement. Ce qui va engendrer une instabilité au niveau du partitionnement du cache. Ceci pénalisera d'un côté le protocole en terme d'efficacité. D'un autre côté, le trafic sera saturé par le nombre des messages de diffusion des mises à jour.

Par ailleurs, la stratégie Round Robin de l'unité d'allocation des blocs de partitionnement est une approche qui vise à équilibrer la distribution des données autour d'un voisinage mais qui ne tient pas compte de la charge du noeud destinataire. En cas du choix d'un voisin à forte charge, le stockage des nouveaux blocs sera donc fait au dépend des données locales de celui-ci. Ce qui engendre naturellement une augmentation des défauts de cache.

De manière plus générale, la stratégie de cache coopératif élastique définit principalement une zone de coopération, où la donnée est autorisée à migrer avec un partitionnement dynamique du cache. Or, dans le cas d'un cas d'un voisinage fortement stressé provoqué précédemment des données locales seront remplacées par les nouveaux blocs partagés ou inversement. Dans des applications où l'on est amené à accéder aux mêmes blocs de données de manière récurrente, cette approche est pénalisantes en nombre de défauts de cache. Les améliorations proposées dans le cadre de ce stage portent principalement sur les trois limitations du mécanisme que nous venons d'analyser.

Le chapitre suivant donne plus de détails sur la contribution, ainsi que le gain en performance qu'apporte le nouveau protocole.

Cinquième partie

Contribution : Mécanisme de glissement de données dans le modèle des caches élastiques

Le mécanisme de cache élastique a pour but principal d'optimiser le partage de l'espace coopératif commun des noeuds d'un voisinage. Comme décrit dans le chapitre précédent, la mise à jour du partitionnement privé/partagé du cache L2 se fait de manière cyclique. Tandis que la stratégie d'allocation des voisins destinataires est basée sur une politique de Round Robin. L'étude d'un scénario de voisinage fortement stressé permet de montrer les limitations de ces deux protocoles d'où la contribution de ce stage. Dans ce chapitre nous présentons deux contributions visant à améliorer le comportement du protocole de cache élastique. La première concerne les mécanismes de partitionnement de cache (privé/partagé). La deuxième porte sur la modification du mécanisme de choix du voisin destinataire.

8 Description du mécanisme de glissement de données

8.1 Politique de remplacement

La première principale contribution consiste à modifier la politique de partitionnement de cache. Elle est basée sur l'utilisation d'un compteur local LHC (Local Hit Counter), incrémenté à chaque accès réussi à la zone privée, et d'un compteur associé à chaque noeud voisin NHC (Neighbor Hit Counter). Chaque NHC est chargé de compter le nombre de fois que le voisin lui étant associé sollicite un accès à l'espace partagé. Les requêtes d'accès mémoire que peut envoyer un noeud demandeur à un son voisin sont de deux types : Accès en lecture/écriture à une donnée stockée dans le cache L2 du voisin. Requête de stockage d'une nouvelle donnée dans le cache L2 du voisin. L'utilisation de ces compteurs permet à chaque noeud d'être en mesure de comparer sa charge locale et celles de ses voisins. Cependant, il n'existe plus de notion de séparation de zones mais il s'agit plutôt d'étiquetage de données selon leurs origines de provenance. Le protocole de cache distingue ainsi entre données privées et données locales. Le partage de cache se fait à la base du taux d'occupation de chacun de ces types de données. Il est de ce fait nécessaire de considérer une politique de remplacement qui permet de décider de la priorité de chaque type de donnée et qui soit fonction de la comparaison des deux types de compteurs. En effet, à l'arrivée d'une requête de stockage d'une donnée trois cas de figure sont à prendre en considération :
1. LHC $>$ somme(NHC) : le noeud effectue plus d'accès à ses données privées que l'ensemble de ses voisins aux donnés partagées. Les données privées sont donc prioritaires. Une nouvelle donnée arrivée remplace la donnée la moins récemment utilisée dans la zone partagée (Shared LRU).
2. LHC $<$ somme(NHC) : l'ensemble du voisinage ou au moins un noeud voisin effectue des accès multiples à la zone partagée. Ceci reflète une charge élevée du voisinage. Le stockage destiné pour la zone partagée se fait donc au dépend des données locales (Private LRU).
3. LHC = somme(NHC) : un troisième cas incarne une forte sollicitation locale. C'est le cas d'accès concurrents aux deux zones du caches L2. Il est difficile de décider de la priorité de remplacement dans une telle situation. Notre contribution participe à ce niveau en proposant la mise en place d'une Politique de remplacement.

Dans l'approche de partitionnement du modèle élastique, ce cas de double sollicitation engendre une forte oscillation entre mode privé et mode partagé. Le cache est donc amené à éjecter les données des

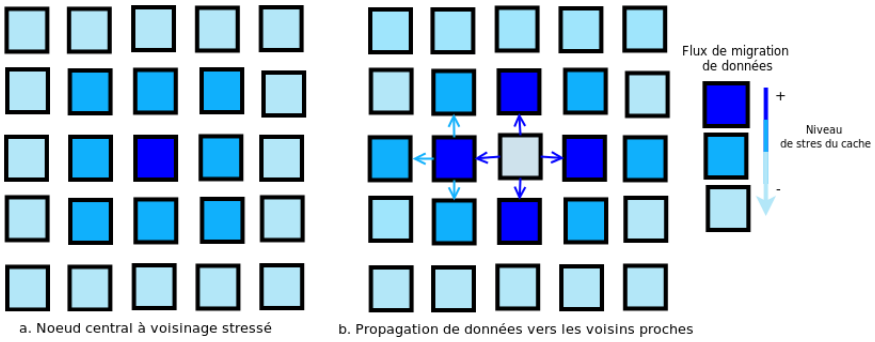

FIGURE 19 – Propagation de données de proche en proche

deux zones à chaque oscillation. Le modèle de remplacement proposé consiste à propager les requêtes de stockage sur un voisinage plus large, tout en respectant le principe de proximité. En effet, l'idée est de faire en sorte que chaque noeud puisse avoir ses données à au plus un pas réseau proche de lui. À chaque requête de stockage d'une donnée voisine, le noeud fait le choix de remplacer une donnée locale. À son tour, cette dernière est envoyée sur son noeud voisin. Le glissement se fait ainsi de proche en proche jusqu'à atteindre un voisinage moins stressé (voir figure 19). Dans cette approche, chaque noeud a une seule chance de migrer vers le voisin (1-Chance forwarding). Cette hypothèse permet d'éviter une large propagation de données, ce qui peut engendrer un goulot d'étranglement dans le réseau. Le glissement est donc limité pour chaque noeud à son voisinage le plus proche.

Ce processus de propagation permet de maintenir les données le plus longtemps possible sur la puce. Étant donné que la réduction des éjections hors puce, réduit le nombre de défauts de cache pour les futurs accès aux données, le glissement se montre efficace en terme d'augmentation du nombre de succès d'accès. Aussi, la proximité réseau des noeuds engendre un gain important en terme de latence d'accès, principalement en cas de voisinage stressé. La politique de remplacement basée sur la comparaison des compteurs évite d'éjecter les données fréquemment utilisées. Elle n'active ainsi le glissement de données de proche en proche que dans le cas de fréquence élevée d'accès aux caches.

8.2 Choix du meilleur voisin

Dans cette section nous présentons une autre approche permettant de décider du voisin destinataire. Dans le mécanisme de cache coopératif élastique l'unité d'allocation des blocs se base sur les données fournies par l'unité de partitionnement, tout en suivant une politique uniforme de Round Robin. Les noeuds stockant le plus de données partagées reçoivent équitablement plus de donnés éjectées du cache local vers le voisinage. Dans une configuration où il faut migrer un grand nombre de données, le Round Robin n'est plus efficace. Les données étant envoyées vers des voisins possiblement plus stressés que d'autres, la politique du choix du meilleur voisin proposée vise à répartir les requêtes de stockage de façon non uniforme sur les différents voisins en prenant en compte la charge de travail de chacun. Ceci est rendu possible grâce à l'utilisation des compteurs. Le meilleur noeud dit Best Neighbor est défini comme étant celui dont le compteur d'accès associé est minimal. Le contrôle de cache , utilise les mêmes données pour décider du bloc à remplacer et de sa destination via un processus de comparaison simple. Cette technique améliore l'efficacité du mécanisme de coopération, en évitant de solliciter les noeuds en pic de charge. Elle évite donc de provoquer des évictions supplémentaires. Le noeud Best Neighbor est finalement celui le plus disponible du voisinage. Comme dans notre protocole de glissement, un noeud ne peut transférer ses données privées qu'à ses voisins proches. Si son compteur associé chez

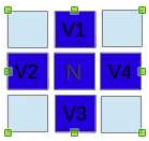

FIGURE 20 – Voisinage d'un noeud N

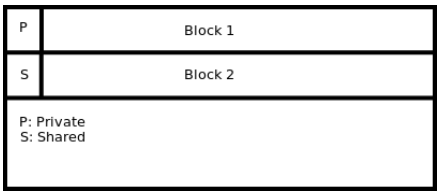

FIGURE 21 – Étiquetage des blocs de données dans le cache L2 d'un noeud

l'un de ses voisins est faible, ceci signifie qu'il n'a pas demandé de l'aide chez ce voisin ou au moins que sa demande d'aide est plus faible par rapport aux autres noeuds. Le gain obtenu avec cette technique par rapport au Round Robin du cache élastique est présenté plus tard dans la partie expérimentale.

8.3 Protocole de glissement

Chaque noeud du maillage forme avec ses 4 voisins adjacents ce que l'on appelle un voisinage (voir figure 20).

Le cache d'un noeud est initialement chargé par ses données privées. Contrairement à la hiérarchie de caches coopératifs élastiques qui définit deux zones différentes de stockage (privée/partagée), la structure du cache dans notre contribution est considérée initialement privée. C'est au moment de sollicitation d'un espace de partage, provenant du voisinage que les données localement stockées, sont remplacées par les nouveaux blocs partagés. Une donnée peut être soit privée (stockée sur son propre noeud), ou partagée (stockée sur un noeud voisin). La figure (voir figure 21) présente l'hypothèse de la structure générale d'un cache L2.

Le noeud cible doit satisfaire toute requête de stockage reçue. La donnée à stocker doit directement être mise à disposition du coeur demandeur pour répondre à sa demande d'accès au plus tôt. Pour ce faire, un espace de stockage flottant est réservé dans tous les caches des noeuds actifs. Le voisin demandeur n'a donc pas besoin d'attendre le retour de tous les noeuds participant au processus de glissement, pour pouvoir envoyer sa donnée. En effet, la dernière donnée éjectée forme l'espace flottant en cours. Le schéma (voir figure 22) décrit de façon détaillée le processus de glissement à 3 pas (i.e le glissement s'arrête à la troisième itération).

9 Implémentation et analyse comparative

9.1 Description fonctionnelle du protocole de glissement de données

Le protocole de glissement de données définit pour chaque noeud deux états de fonctionnement : Un premier état AVAILABLE où le cache est disponible, la donnée arrivée est directement stockée dans la mémoire et étiquetée selon sa provenance (privée/partagée). Le passage à l'état suivant à savoir l'activation du glissement SLIDING est conditionné par la saturation de l'espace de stockage du cache

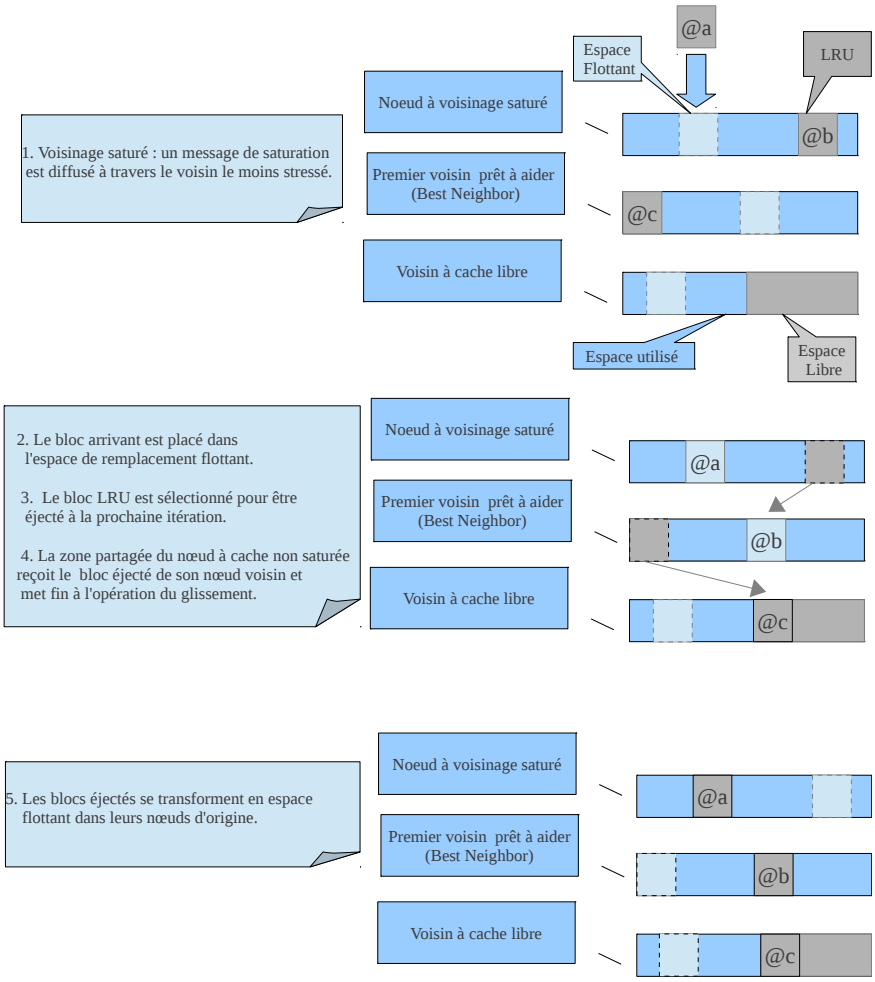

FIGURE 22 – Description d'un processus de glissement à 3 pas

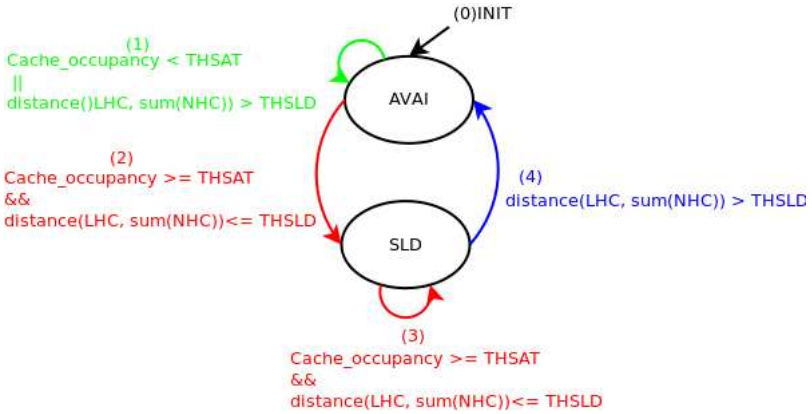

FIGURE 23 – Machine d'état du protocole élastique avec glissement

L2 local, mais aussi par le franchissement du seuil de stress du voisinage. En effet, la saturation d'un noeud revient à remplir son cache L2. Le passage à l'état de stress est fonction des compteurs définis précédemment. Un noeud est dit stressé si la distance entre son compteur local (LHC) et la somme des compteurs associés à ses voisins (NHC) est inférieure à un certain seuil noté TH_SLD, que l'on définit empiriquement. L'intérêt du seuillage est de permettre un aspect adaptatif pour la gestion du stress en fonction de la taille du cache et du type d'applications utilisées.

Si la charge des caches baisse, on initialise les compteurs et on revient à l'état disponible. La machine d'état présentée dans la figure (Voir figure 23) illustre la description fonctionnelle du mécanisme de glissement.

9.2 Plate-forme de test

Nous utilisons une étude analytique pour l'évaluation du protocole de glissement proposé. L'approche analytique se base sur la génération de statistiques concernant la charge du trafic réseau en fonction de la nature des messages circulant et des noeuds communicant. Une configuration du modèle étudié est définie à partir de l'architecture de la puce et de types de communications établies entre les différents coeurs. Ce même modèle est utilisé pour implémenter les différents protocoles de cohérence étudiés :

- Le protocole Baseline ;
- Le protocole Coopératif élastique ;
- Le protocole Coopératif par glissement de données

L'analyse porte sur l'observation des comportements des différents protocoles en comparant les statistiques obtenues pour chacun. Le Cache Validator , est l'outil de validation utilisé pour implémenter les tests. Le Cache Validator, développé au CEA, lit à partir d'un fichier d'entrée une suite d'accès mémoire. Il permet de générer à partir de ces accès mémoire, de l'architecture de la puce et du protocole sélectionné, toutes les statistiques sur le trafic réseau. Le fichier d'entrée du Cache Validator est constitué d'un ensemble de lignes, chacune comportant des informations sur l'adresse et la taille de la donnée, le type d'accès ainsi que l'identifiant du noeud demandeur (voir figure 24).

Nous considérons une architecture en maille à 16 coeurs. Les communications sur puce se font en mode point à point. Différents types de messages sont définis dans l'outil de validation. Ils sont classés suivant leur noeud destinataire, et leur type de requête associé.

FIGURE 24 – Format de trace d'accès mémoire

Messages destinés au Home Node : différentes requêtes sont envoyées au Home Node ; lecture (RD_RQ_TO_HN), écriture (WR_RQ_TO_HN), invalidation (INV_RQ) et mise à jour de l'état de la donnée (DOWN_RQ).
Message destiné au noeud voisin : les premiers tests utilisent un seul type de requête vers le voisin (MSG_RQ_TO_NGH). Un noeud demandeur contacte le Home Node pour satisfaire les défauts d'accès à son cache local. Nous considérons dans l'analyse du coût de ces accès, l'ensemble des communications point à point engendrées par le chemin d'accès suivant une distance de Manhattan. L'accès à une donnée sur le noeud voisin étant directe, représente ainsi un coût moins élevé par rapport au premier (voir figure 25).

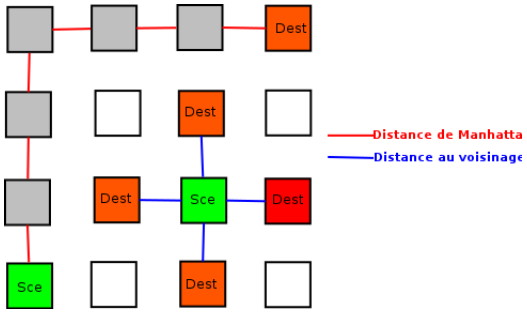

FIGURE 25 – Distance de Manhattan versus communication au voisinage

Les paragraphes suivants présentent les différentes configurations testées ainsi que les résultats obtenus. L'analyse des résultats issues des différents scénarios de tests se base sur cette différence du coût d'accès aux données.

10 Premiers tests et analyse de performances du mécanisme de glissement

10.1 Évaluation du trafic par glissement de données

Analyse du scénario : La principale motivation de la contribution est de réduire le nombre de défauts de cache sur un voisinage stressé. Dans ce premier test nous étudions un scénario où le cache est fortement sollicité. Les accès privés et partagés se succèdent de façon alternative entre le noeud central et son voisinage. Nous rappelons que nous étudions le cas de mémoire à une ligne de cache. Dans une première phase les noeuds voisins sont saturés par les accès aux données B (Nord), C (Ouest), D (Sud), E (Est). Dans une deuxième phase le noeud central effectue des accès successifs au lot de données A1..A5 (voir figure 26). En alternant ces deux phases d'accès aux caches, tout le voisinage se trouve dans un état de stress. Ce qui favorise la migration des données dans une approche coopérative. Le noeud central aura une fréquence d'accès aux zones partagées de ses voisins plus importante. Dans un mécanisme de partitionnement élastique, les données des voisins sont éjectées afin de stocker les

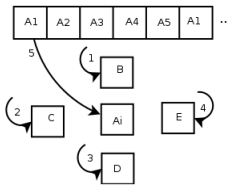

FIGURE 26 – Schéma du scénario d'accès aux données

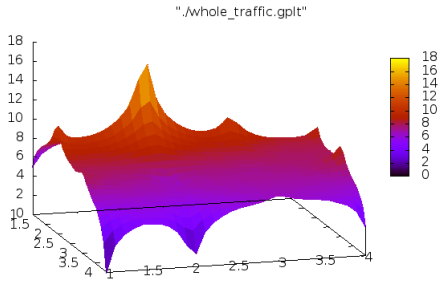

FIGURE 27 – Trafic résultant d'un protocole élastiques

blocs en provenances du noeud central. Cependant, le protocole de glissement permet aux voisins de transférer leurs données locales à leurs voisinages respectifs au lieu de les éjecter hors puce.

Observation des résultats en nombre de messages : L'observation des résultats obtenus, montre que le nombre de messages qui circulent sur le réseau est relativement proche pour les deux protocoles, avec une réduction de la zone de communication dans le mode glissement. Par contre, on constate que le mode glissement augmente la charge du voisinage (voir figures 27, 28).

Par ailleurs, si on compare avec les résultats du protocole Baseline, nous constatons que le trafic est fortement réduit au niveau de la maille (zone de communication) ; en effet, il s'agit des communications point à point vers les Home Nodes, qui nécessitent l'implication des autres noeuds. Ceci est donc traduit par la diminution de l'activité des Home Nodes dans les deux modes élastique coopératif et élastique

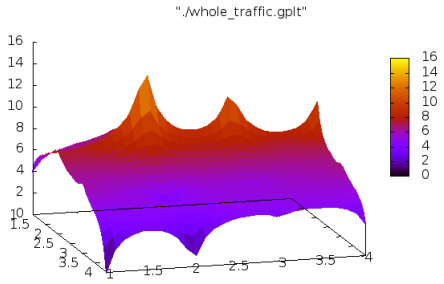

FIGURE 28 – Trafic par glissement de données

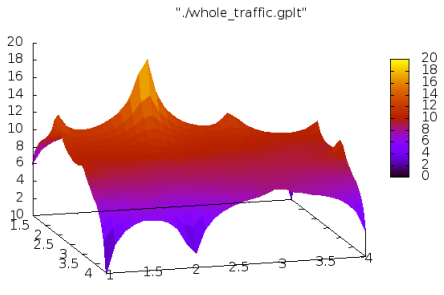

FIGURE 29 – Trafic pour le protocole Baseline

par glissement par rapport au protocole Baseline simple, et donc une diminution du trafic global (voir figure 29).

Analyse des observations : Au niveau du protocole élastique simple les défauts de cache sont traduit par l'augmentation du nombre de messages vers le Home Node (assimiler au coût d'accès en mémoire externe). Si on le compare avec un protocole Baseline, nous allons constater que le trafic est remarquablement réduit grâce au processus de coopération classique qui nous réduit les défauts de cache , et donc les accès aux home Nodes. D'autre part, la réduction du nombre de messages dans le protocole de glissement n'est pas la bonne approche pour estimer l'efficacité de ce mécanisme. En effet, la propagation de la migration des données sur le voisinage, maintient naturellement la charge du trafic à cause du nombre de messages échangés entre voisins. Or, il est important de considérer l'aspect de proximité. L'accès à une données localisée à un pas est toutefois moins pénalisant qu'un accès distant (accès au Home Node par exemple). Le troisième constat qui consiste en la réduction de la zone de propagation des messages sur la maille, montre que :

- Grâce à la coopération élastique nous limitons les accès Home Node (défauts de cache), car les données sont maintenues au voisinage du noeud central ;
- La zone de communication est encore mieux réduite dans une approche par glissement, grâce à la réduction des éjections de données notamment celles des données des voisins du noeud central

10.2 Le choix du meilleur voisin

Analyse du scénario : Le scénario de test pour évaluer la technique du choix du meilleur voisin consiste à stresser un noeud central, ainsi que deux de ses voisins (Nord et Ouest)(voir figure 30). Nous chargeons les données B et C respectivement dans les caches de ces voisins. Ensuite le noeud centrale accède successivement au lot données A1, A2, A3. La charge du noeud central active un processus coopératif, qui consiste à solliciter les caches voisins. Une politique de Round Robin, va distribuer les données du noeud central, de manière équilibrée, sur tout le voisinage. Les données privées des deux voisins (Nord et Ouest) sont éjectées hors puce. De plus, ces deux voisins appellent périodiquement leurs données privées. Il en résulte que les données du noeud central stockées chez ces deux voisins (Nord et Ouest) sont également perdues. Cependant, la politique du choix du meilleur voisin basée sur la comparaison des compteurs de tout le voisinage prend en compte le besoin en mémoire de ces deux noeuds. Ceux-ci sont moins sollicités et leurs données privées sont plus protégées. Il en est de même pour le noeud central. Ce test montre principalement la distribution des échanges du noeud central

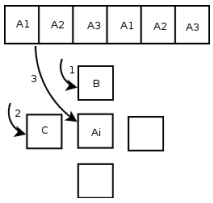

FIGURE 30 – Schéma du scénario d'accès aux données pour le Best Neighbor

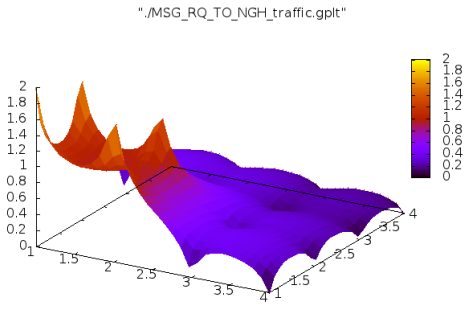

FIGURE 31 – Le flux de communication dans un voisinage avec la politique Round Robin

avec ses voisins, et le nombre de défauts de cache pour chaque noeud.

Observation des résultats en nombre de messages : Nous observons dans un premier le flux de communication du noeud central et ses voisins. Nous constatons que dans le protocole élastique où le choix du voisin destinataire se fait grâce à une politique Round Robin, les voisins sollicités pour le stockage des données du noeud stressé sont le noeud Nord et Ouest (voir figure 31).

En appliquant la politique du meilleur voisin, les transferts de données du noeud central sont plutÃ´t orientés vers les deux autres voisins à savoir le noeud Sud et Est (voir figure 32).

Concernant le coût d'accès en mémoire externe, nous observons le nombre de requêtes RD_RQ_TO_HN envoyées vers le Home Node par chaque noeud :
Le noeud central : représente le nombre maximal d'accès aux Home Nodes. Cette valeur passe de 6 à 3

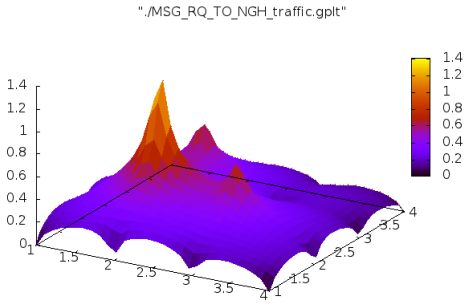

FIGURE 32 – Le flux de communication dans un voisinage avec la politique Best Neighbor

entre les deux protocoles ce qui correspond à un gain de 50% en coût d'accès Home Node (voir figures 33, 34).
Pour les deux noeuds voisins : il était de même en terme d'accès aux Home Nodes. Le nombre de messages est réduit de moitié grâce au mécanisme du Best Neighbor (suppression de 2 requêtes/noeud pour 4 accès en cache).

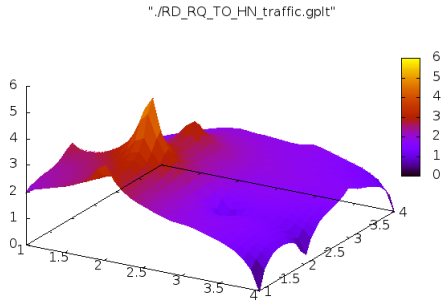

FIGURE 33 – Requêtes vers Home Node dans une approche Round Robin

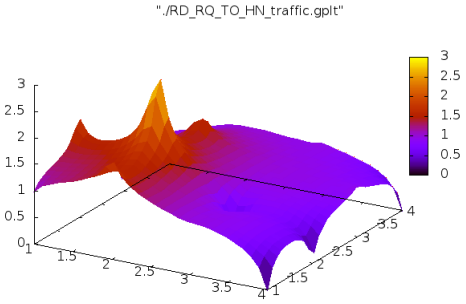

FIGURE 34 – Requêtes vers Home Node dans une approche Best Neighbor

Analyse des observations :
Deux conclusions se tirent de ces résultats :

Le choix du voisin est plus pertinent : Nous avons remarqué que grâce à l'approche Best Neighbor le noeud central choisit d'envoyer ses requêtes vers les voisins Sud et Ouest. Car, contrairement à la technique Round Robin, ce choix évite de stocker les données chez les noeuds saturés au dépend de leurs propres données.

Le nombre de défauts de cache est réduit : Le choix du meilleur voisin protège les données privées des deux voisins saturés, mais aussi les données partagées du noeud central. En effet, en alternant les accès aux données entre noeud central et les deux voisins cibles (Nord et Ouest), ces derniers éjectent, une première fois, leurs données privées pour les remplacer par les données partagées du noeud central et une deuxième fois les données partagées pour stocker leurs données locales. Ce comportement instable explique l'augmentation du nombre de requêtes vers le Home Node (2 fois plus grand pour le Round Robin) qui est traduit dans notre analyse par des défauts d'accès au cache local

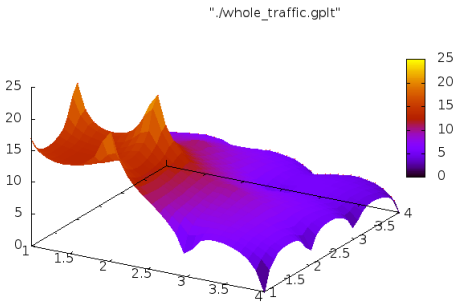

FIGURE 35 – Trafic du voisinage stressé avec politique de partitionnement cyclique

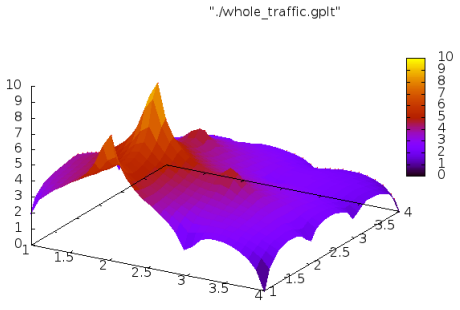

FIGURE 36 – Trafic du voisinage stressé avec remplacement prioritaire

de chaque noeud.

10.3 Efficacité de la politique de remplacement par priorité

Analyse du scénario :
Ce troisième scénario évalue la robustesse de la stratégie de partitionnement du cache élastique dans des situation de voisinage fortement stressé. Particulièrement, en ce qui concerne le partitionnement basé sur un nombre de cycle processeur qui permet d'adapter le partage de la mémoire cache aux besoins de l'application. Il s'agit de faire une comparaison de cette approche vis à vis du concept de remplacement par priorité qui abandonne la notion de mise à jour cyclique. Nous considérons des mémoires à deux lignes de cache. Les accès sont répartis de manière à saturer tout le voisinage dans un premier temps. Un stress général est ensuite provoqué au niveau de tous les noeuds du voisinage via la multiplication de leurs accès mémoire.

Observation des résultats en nombre de messages :
L'observation générale porte sur le nombre totale des messages qui circulent sur le réseau. La valeur maximale du trafic que présentent les pics au niveau des noeuds stressés, passe de 25 à 10 avec implémentation du mécanisme de remplacement prioritaire. Une vison globale des statistiques obtenues montre que le trafic est remarquablement réduit ($>$ 50 % de messages éliminés) (voir figures 35, 36).

Analyse des observations :

Dans le cas où l'ensemble des noeuds d'un voisinage est stressé, un grand nombre de requêtes vers le Home Node est envoyé à travers les différents noeuds faisant partie de ce voisinage. En effet, la réduction de ce nombre lors du passage dans le mode coopératif par glissement est expliqué par l'aspect adaptatif amélioré de ce mode. Alors que le mode de partitionnement élastique consiste à mettre à jour le partage de la mémoire tous les N cycles processeur, et à remplacer les données suivant le partitionnement courant, notre contribution se base sur une prise de décision instantanée concernant le bloc à remplacer sans avoir à considérer la frontière entre les zones privée/partagée. L'aspect adaptatif du remplacement par priorité se montre plus robuste vis à vis de la montée du besoin en mémoire de l'application. Il permet donc de mieux gérer le stockage de données dans les caches partagés afin de réduire les éjections hors puce et de limiter par conséquence les défauts de cache.

Sixième partie

Conclusion

L'objectif de ce stage est de proposer un mécanisme de cohérence de cache destiné aux architectures manycoeurs. L'état de l'art effectué a permis d'étudier l'approche des caches coopératifs élastiques. Les limitations de cette technique en cas d'augmentation du besoin en mémoire, a conduit à proposer la technique de glissement de données. Cette contribution apporte une amélioration sur les deux aspects du cache coopératif ; à savoir le choix du meilleur voisin et la politique de partage du cache. L'idée repose sur deux principaux points :

1. **Utilisation de compteurs de voisinage** : utiliser sur chaque noeud un compteur d'accès local LHC et un ensemble de compteurs associés au voisins directs NHC. Toute prise de décision pour le remplacement d'un bloc dépend de ces compteurs ;
2. **Autorisation de la propagation de données de voisin en voisin** : Cette technique permet de répartir la charge de stockage de manière équilibrée et de réduire le nombre de défauts de cache. Chaque noeud cède ses données locales à son voisin proche pour libérer de l'espace aux données partagées. Cette approche permet d'établir un compromis entre une utilisation équilibrée des caches sur la puce et la proximité des données lorsqu'elles sont stockées sur des noeuds différents. Les résultat montrent que les temps d'accès moyens dans une configuration de forte sollicitation sont inférieurs aux autres stratégies.

Dans l'approche du cache élastique, une première unité chargée du partitionnement de cache , envoie à chaque mise à jour les informations du partage en zone privée/partagée à l'unité d'allocation des blocs. Cette dernière s'occupe d'allouer en Round Robin un noeud destinataire à la donnée migrant sur le voisinage. Cependant, les nouvelles techniques de remplacement et du choix du meilleur voisin proposées par la contribution, permettent de simplifier la structure noeud à cache élastique. Une seule unité de contrôle de cache peut gérer le stockage des données reçues par un noeud. Cette simplification élimine, d'un côté, le trafic des métadonnées, généré par la communication entre l'unité de partitionnement et l'unité d'allocation de blocs. Elle permet également de réduire le coût et la consommation en énergie.

Pour des futurs améliorations du mécanisme, il est prévu d'étudier l'accès aux données stockées en mode partagé par leur noeud accueillant. C'est une perspective qui pourrait améliorer l'efficacité du mécanisme en terme de temps d'accès et renforcer la coopération à l'intérieur du voisinage.

Au niveau, des tests de performances, il est prévu d'effectuer des expérimentations en implémentant des traces d'applications à grand besoin en mémoire (exp : Traitement d'image) sur une plateforme de simulation. Toutefois, dans une perspective plus poussée, il est envisageable de réaliser une implémentation sur puce matérielle.

Table des figures

Références

[1] E. Herrero, J. González, and R. Canal, "Power-efficient spilling techniques for chip multiprocessors.," in *Proceedings of the 16th international Euro-Par conference on Parallel processing : Part I*, pp. 256–267, 2010.

[2] E. Herrero, J. González, and R. Canal, "Elastic cooperative caching : an autonomous dynamically adaptive memory hierarchy for chip multiprocessors," in *ISCA*, pp. 419–428, 2010.

[3] J. K. Bennett, J. B. Carter, and W. Zwaenepoel, "Munin : Distributed shared memory based on type-specific memory coherence.," in *Proceedings of the second ACM SIGPLAN symposium on Principles & practice of parallel programming*, pp. 168–176, 1990.

[4] S. T. Jhang, "A new write-invalidate snooping cache coherence protocol for split transaction bus-based multiprocessor systems.," in *TENCON'93. Proceedings. Computer, Communication, Control and Power Engineering*, pp. 229–232, 1993.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *SOSP'03*, pp. 29–43, 2003.

[6] J. S. Gwertzman and M. Seltzer, "The case for geographical push-caching," in *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V).*, p. 51, 1995.

[7] K. Li, "Ivy : A shared virtual memory system for parallel computing.," in *ICPP (2)'88*, pp. 94–101, 1988.

[8] B. B. N., Z. M. J., and S. W. A., "The midway distributed shared memory system.," tech. rep., 1993.

[9] Z. Yuanyuan, I. Liviu, and L. Kai, "Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems.," in *Proceedings of the second USENIX symposium on Operating systems design and implementation*, pp. 75–88, 1996.

[10] L. Iftode, J. P. Singh, and K. Li, "Scope consistency : A bridge between release consistency and entry consistency.," in *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pp. 277–287, 1996.

[11] K. John, B. David, C. Yan, C. Steven, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "Oceanstore : an architecture for global-scale persistent storage.," in *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pp. 190–201, 2000.

[12] E. Herrero, J. González, and R. Canal, "Ivy : a read/write peer-to-peer file system.," in *Proceedings of the 5th symposium on Operating systems design and implementation*, pp. 31–44, 2002.

[13] S. Ion, M. Robert, K. David, K. M. Frans, and B. Hari, "Chord : A scalable peer-to-peer lookup service for internet applications.," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 149–160, 2001.

[14] R. Antony and D. Peter, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility.," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 188–201, 2001.

[15] R. A. I. T. and D. Peter, "Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems.," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pp. 329–350, 2001.

[16] J.-M. Busca, F. Picconi, and P. Sens, "Pastis : a highly-scalable multi-user peer-to-peer file system.," in *Proceedings of the 11th international Euro-Par conference on Parallel Processing*, pp. 1173–1182, 2005.

[17] J. Cho, S. Oh, J. Kim, H. H. Lee, and J. Lee, "Neighbor caching in multi hop wireless ad hoc networks.," *Communications Letters, IEEE*, pp. 525–527, 2003.

[18] N. Chand, R. C. Joshi, and M. Misra, "Cooperative caching in mobile ad hoc networks based on data utility," *Mob. Inf. Syst.*, pp. 19–37, 2007.

[19] N. Chand, R. C. Joshi, and M. Misra, "Cooperative caching in mobile ad hoc networks based on data utility," *Mobile Information Systems*, pp. 19–37, 2007.

[20] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. L. Hennessy, "The directory-based cache coherence protocol for the dash multiprocessor.," in *International Symposium on Computer Architecture*, pp. 148–159, 1990.

[21] L. Cheng, J. B. Carter, and D. Dai, "An adaptive cache coherence protocol optimized for producer-consumer sharing.," in *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pp. 328–339, 2007.

[22] M. J. F., T. Josep, and D. José, "Improving the performance of bristled cc-numa systems using virtual channels and adaptivity.," in *Proceedings of the 13th international conference on Supercomputing*, pp. 202–209, 1999.

[23] C. K. Tang, "Cache system design in the tightly coupled multiprocessor system," in *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pp. 749–753, 1976.

[24] L. M. Censier and P. Feautrier, "A new solution to coherence problems in multicache systems.," *IEEE Trans. Comput.*, pp. 1112–1118, 1978.

[25] K. Gharachorloo, L. A. Barroso, and A. Nowatzyk, "Efficient ecc-based directory implementations for scalable multiprocessors," 2000.

[26] A. Ros, M. E. Acacio, and J. M. GarcÃa, "A direct coherence protocol for many-core chip multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, pp. 1779–1792, 2010.

[27] K. Strauss, X. Shen, and J. Torrellas, "Flexible snooping : Adaptive forwarding and filtering of snoops in embedded-ring multiprocessors," in *ISCA'06*, pp. 327–338, 2006.

[28] D. J. Sorin, M. Plakal, A. Condon, M. D. Hill, M. M. K. Martin, and D. A. Wood, "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol," *IEEE Trans. Parallel Distrib. Syst.*, pp. 556–578, 2002.

[29] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in *ISCA'06*, pp. 264–276, 2006.

[30] M. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching : Using remote client memory to improve file system performance.," in *Operating System Design and Integration*, pp. 267–280, 1994.

[31] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors.," in *International Conference on Supercomputing*, pp. 242–252, 2007.

[32] E. Herrero, J. González, and R. Canal., "Distributed cooperative caching.," in *Parallel Architectures and Compilation Techniques*, pp. 134–143, 2008.

[33] E. Herrero, J. González, and R. Canal, "Distributed cooperative caching : An energy efficient memory scheme for chip multiprocessors.," *IEEE Trans. Parallel Distrib. Syst.*, pp. 853–861, 2012.

[34] P.Kuppusamy, Dr.K.Thirunavukkarasu, and Dr.B.Kalaavathi, "A review of cooperative caching strategies in mobile ad hoc networks.," *International Journal of Computer Applications*, pp. 22–26, 2011.